Interpretability-driven Intelligent Software Reliability Engineering

HE, Shilin

A Thesis Submitted in Partial Fulfilment
of the Requirements for the Degree of
Doctor of Philosophy
in
Computer Science and Engineering

The Chinese University of Hong Kong
August 2020

Thesis Assessment Committee

Professor LO Chi Lik Eric (Chair)

Professor LYU Rung Tsong Michael (Thesis Supervisor)

Professor XU Qiang (Committee Member)

Professor KEUNG Wai Jacky (External Examiner)

Abstract of thesis entitled:

Interpretability-driven Intelligent Software Reliability Engineering

Submitted by HE, Shilin

for the degree of Doctor of Philosophy

at The Chinese University of Hong Kong in August 2020

The reliability of software, including traditional software and intelligent software, is crucial to both end-users and service providers. However, the increasing scale and complexity of traditional software and the massive parameters of intelligent software, make both software hard to understand, which poses great challenges for software reliability engineering. Especially, detecting and tracking the problems becomes inefficient and error-prone when unexpected software behaviors occur. In this thesis, we study the intelligent software reliability engineering from the interpretability perspective. Specifically, we propose to interpret traditional software with logs and intelligent software with gradients and bilingual knowledge.

Firstly, we conduct an experience report on log-based anomaly detection, which uncovers abnormal software behaviors using the interpretable logs. The anomaly detection plays a vital role in traditional software reliability engineering. Although anomaly detection has been widely studied in recent years, a comprehensive benchmark and an open-source toolkit are lacking. We implement six representative anomaly detection

methods and evaluate their performance in terms of effectiveness and efficiency. We obtain five insightful findings and make these methods open-source for easy reuse and further study.

Secondly, we propose an intelligent log-based method to identify impactful problems in software systems. There could be various types of problems, while some are more impactful, leading to the degradation of Key Performance Indicator (KPI). To tackle the challenges of highly imbalanced log distribution and label scarcity, we propose a novel cascading clustering and KPI correlation method to identify impactful problems. Experimental results on three real-world datasets confirm the effectiveness and efficiency of our proposed method.

Thirdly, we focus on interpreting the intelligent software, which is the first step towards its reliability engineering. We take the neural machine translation (NMT) model as our testbed. Although NMT advances the state-of-the-art performance, its unsatisfactory interpretability poses great challenges for model debugging and improving. We propose to understand the model input-output behaviors by exploiting the internal gradients to estimate the word importance. Comprehensive experiments on different perturbation operations, language pairs, and model architectures validate the effectiveness of our method. We also employ the proposed word importance to detect the under-translation error in NMT.

Fourthly, we explore the interpretability of NMT models by assessing its learned bilingual knowledge. The NMT model contains tons of uninterpretable parameters, but the essential bilingual knowledge that model embeds for translation is unclear. We propose to quantitatively assess the bilingual knowledge that NMT models learn using the human-understandable phrase

table. Extensive experiments on widely-used datasets show that the phrase table is reasonable and consistent. Moreover, we obtain some interesting findings after analyzing the model learning dynamics and some advanced model improvement techniques.

In summary, this thesis targets at studying the interpretability-based reliability engineering of both traditional software and intelligent software. Comprehensive experiments on widely-used datasets confirm the effectiveness and efficiency of our proposed methods.

論文題目：解釋性驅動的智能軟件可靠性工程
作者　　：何世林
學校　　：香港中文大學
學系　　：計算機科學與工程學系
修讀學位：哲學博士
摘要　　：
軟件，包括傳統軟件和智能軟件，其可靠性對於終端用戶和服務提供方都至關重要。然而，傳統軟件急劇增加的規模和複雜度，以及智能軟件中巨量的參數，使得軟件難以被理解，從而給軟件可靠性工程帶來了巨大的挑戰。尤其當意料之外的軟件行為出現的時候，檢測和追蹤問題變得非常的低效和易錯。在本論文中，我們從可解釋性的角度探究智能軟件可靠性工程。具體來說，我們提出用日誌來解釋傳統軟件以及用梯度和知識來解釋智能軟件。
首先，我們基於日誌的異常檢測算法基礎上進行經驗研究。異常檢測在傳統軟件可靠性分析中極為重要，其目的在於利用可解釋性的日誌來找出異常的軟件行為。雖然異常檢測已被廣泛研究，但始終缺乏詳盡的基準探究和開源工具。我們實現了六種代表性的異常檢測方法，並且從有效性和效率的角度對這些方法進行評估。我們總結出五個有趣結論，並將這些方法開源以供後續使用和研究。
其次，我們提出一種智能化的，基於日誌的方法來鑒別出軟件系統中有影響力的問題。軟件系統中的問題種類可能有很多，但是有些問題會相較於其餘問題更有影響力，也更容易導致系統性能監控指標的下降。為了解決日誌分佈極度不均衡和標籤

缺乏的挑戰，我們提出一種新穎的級聯聚類以及關聯系統性能監控指標的方法來鑒別出有影響力的問題。我們在三個真實數據集上驗證了我們方法的有效性和高效率。

再其次，我們關注與解釋智能軟件的行為，該可解釋性是邁向智能軟件可靠性工程的第一步。在本論文中，我們將神經機器翻譯模型作為我們的研究對象。雖然神經機器翻譯模型已經極大地促進了機器翻譯任務的效果，它的不可解釋性卻阻礙了模型的糾錯和提升。我們提出用模型內部的梯度信息來估計詞重要性，從而理解模型的輸入-輸出行為。我們在不同擾動，語言對和模型架構上實驗證明我們方法的有效性。我們還將這種基於梯度的方法用於檢測神經機器翻譯的漏譯問題。

最後，我們從評估模型學習到的雙語知識角度來探究神經機器翻譯模型的可解釋性。神經機器翻譯模型中包含大量不可解釋的參數，並且模型中的翻譯所需雙語知識是不清楚的。我們提出用短語詞表的方法來量化評估模型學習到的雙語知識。我們在常見數據集上進行大量的實驗，結果表明短語表是一種合理且通用的評估方法。除此之外，我們分析了模型的動態學習過程和一些模型改進方法，並從中得到了一些有趣的結論。

綜上所述，本論文旨在從可解釋性角度出發研究傳統軟件和智能軟件的可靠性工程。我們在廣泛使用的數據集上進行了大量豐富的實驗，結果證明了我們所提出的這些方法的有效性和高效性。

Acknowledgement

First and foremost, I would like to thank my supervisors, Prof. Michael R. Lyu, for his excellent supervision during my Ph.D. study at CUHK. From choosing the research topic to technical writing, his inspiring guidance and patience help me conduct tough research work. During the long Ph.D. study period, I have learned so much from his knowledge and attitude in doing research.

I am grateful to my thesis assessment committee members, Prof. LO Chi Lik Eric and Prof. XU Qiang, for their constructive comments and valuable suggestions to this thesis and all my term presentations. Great thanks to Prof. KEUNG Wai Jacky from City University of Hong Kong, who kindly serves as the external examiner for this thesis.

I would like to thank my mentor, Mr. Qingwei Lin, when I interned in Microsoft Research Asia for the insightful discussions on log-based problem identification. Also, I would like to thank Dr. Zhaopeng Tu and Dr. Xing Wang when I interned in Tencent AI Lab for their valuable contributions to the research in this thesis.

I would like to thank Dr. Pinjia He and Dr. Jieming Zhu for their insightful discussion and inspiring guidance on the research topic in the early stage of my Ph.D. study.

I am also thankful to my fantastic group fellows, Yu Kang,

Hongyi Zhang, Pinjia He, Tong Zhao, Xiaotian Yu, Hui Xu, Cuiyun Gao, Jichuan Zeng, Jiani Zhang, Ken Chan, Jian Li, Han Shao, Wang Chen, Yue Wang, Pengpeng Liu, Haoli Bai, Wenxiang Jiao, Yifan Gao, Jingjing Li, Weibin Wu, Zhuangbin Chen, Tianyi Yang, Wenchao Gu, and Jen-Tse Huang.

Last but most important, I would like to thank my parents. Their deep love and constant support are the driving force during my Ph.D. study.

To my family.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis presents our research on interpretability-driven intelligent software reliability engineering, an important field in software engineering. We first provide a brief overview of the research problems understudy in Section 1.1 and highlight the main contributions of this thesis in Section 1.2. Section 1.3 outlines the thesis structure.

1.1 Overview

With the revolution of computer techniques and the increase of user demands, the software has been providing the commercial and social infrastructure of modern life to everyone, such as communication, e-commence, and travelling. In general, the software includes traditional software (Software 1.0, e.g., standalone software, distributed systems, service systems) and intelligent software (Software 2.0, e.g., artificial intelligence applications). The software, including search engines such as Google Search, social networks such as Facebook, and machine translators like Google Translate, brings excellent convenience to human-beings and increases human productivity tremendously

by providing services to millions of end-users around the world. The software reliability is crucial to both end-users and software providers since a tiny error might lead to user dissatisfaction, revenue loss, and even human life in danger. As an example, in traditional software, Amazon Web Service (AWS) encountered an outage problem in its simple storage service (S3) in 2017[1], which breaks down a lot of websites and applications built upon it, and millions of users are affected. In terms of intelligent software, in June 2020, a Tesla car, which is on the autopilot mode, crashes into an overturned truck on a busy highway in Taiwan[2], demonstrating that the unreliable intelligent software might threat human lives or even lead to more severe consequences.

However, the increasing complexity and scale of software nowadays make the software hard to understand, which poses significant challenges to the problem detection and troubleshooting and thereby hinders the achieving of software reliability. Notably, traditional software such as the system software contains millions of lines of source code distributed in a large number of strongly-coupled components and modules, making it notoriously hard to comprehend the software, let alone debugging and troubleshooting. For example, Spark [2] contains around 1.32 million lines of source code, and Hadoop [1] contains 4.10 million lines of source code[3]. Intelligent software such as the Transformer [148] consists of tens of millions of uninterpretable real-value parameters. Under such a circumstance, it is nec-

[1]https://techcrunch.com/2017/02/28/amazon-aws-s3-outage-is-breaking-things-for-a-lot-of-websites-and-apps/]

[2]https://www.forbes.com/sites/bradtempleton/2020/06/02/tesla-in-taiwan-crashes-directly-into-overturned-truck-ignores-pedestrian-with-autopilot-on

[3]Online statistics accessed in June 2020

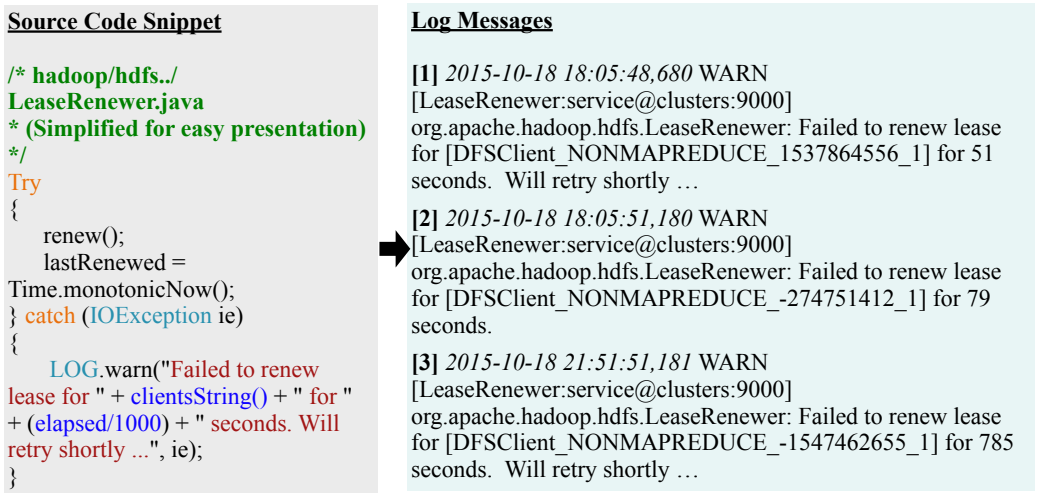


Figure 1.1: An example of a Hadoop code snippet and its generated log messages.

essary for both developers and maintainers to understand the logic behind the software before completely fixing the potential bugs [124, 51, 136]. Therefore, the interpretability of software becomes a must for software reliability engineering.

In traditional software, to interpret program executions and debug unexpected behaviors, developers often resort to program analysis methods such as testing or debugger tools (e.g., Java Debugger). However, this method is very inefficient and does not apply to large-scale distributed systems. As an alternative, a typical solution is to leverage the software logs. Unlike the source code, logs are mostly in natural language and describe a specific program event in a human-understandable and abstractive manner. As shown in Figure 1.1, the left part is a code snippet of the try-catch block extracted from the Hadoop system, in which the logging statement describes the detailed information about the exception. The right part shows three log messages generated from the logging statement.

In large-scale software, developers insert logging statements to record the detailed information of program execution flows (such as when processing a user request), which by natural can interpret the software behaviors and help detect the software problems. As a common practice, developers usually inspect the logs manually or use heuristic rules to understand software behaviors. For example, they extract the related logs by searching keywords (e.g., "fail", "exception") in log messages. However, these human-centered methods are inadequate for large-scale software because of the following reasons:

- Large-scale software generates tons of logs, for example, at about 50 gigabytes (around 120~200 million lines) per hour [101]. The sheer volume of logs makes it notoriously difficult, if not infeasible, to clearly interpret the software behaviors and discern the key problem information from the log data manually.

- The large-scale, parallel nature of software such as distributed system makes its behaviors too complex to comprehend by a single developer, who is often responsible for sub-components only. For example, many systems (e.g., Hadoop, Spark) are implemented by hundreds of developers. A developer might have only an incomplete understanding of overall software behaviors, making it challenging to identify problems from massive logs.

- The fault tolerance mechanism in distributed systems makes the logs redundantly generated and further disturb the software understanding. Traditional methods such as keyword search become ineffective in extracting suspicious log messages and likely lead to many false positives. The

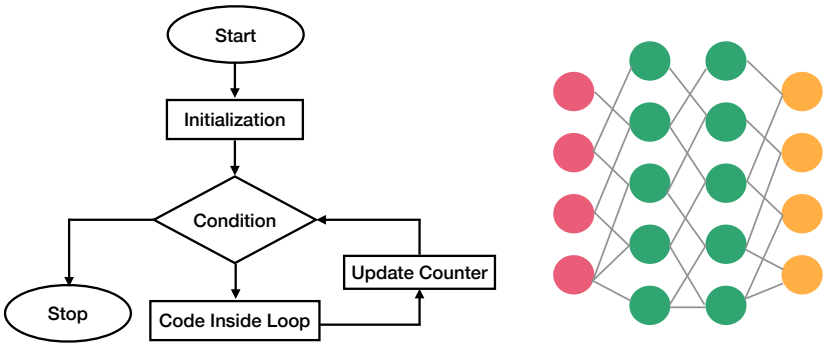
identified logs might be unrelated to real failures [90], which will significantly increase the manual inspection effort.

To tackle the above challenges, in this thesis, we resort to intelligent methods (i.e., machine learning techniques) for automated log analysis in software reliability engineering. Specifically, we focus on two closely-related classification tasks, anomaly detection, and problem identification. The task details are listed as follows:

Anomaly detection: The task aims to detect abnormal software behaviors and distinguish them from their counterparts. These anomalies are very likely to be software problems.

Problem identification: The task not only distinguishes the software problems from normal software behaviors but also identifies different types of problems using the log data.

Recent years have also witnessed the rapid rise and wide adoption of intelligent software in many areas, such as speech recognition in Siri, machine translation in Google Translate, face identification in phone unlocking. Similar to the traditional software, intelligent software provides services (Machine Learning as a Service, MLaaS) to end-users, and its reliability is also crucial. In recent studies [59, 6, 166], intelligent software such as deep learning models are often criticized for their sensitivity and vulnerability. When adding some human-imperceptible noises or perturbations (i.e., adversarial examples), the models are prone to make wrong predictions. For example, after adding some background noises, the speech of "How are you?" can be incorrectly identified as "Open the door." Besides, some regular examples may also confuse the model to make wrong predictions, e.g., corner cases that are not learned well in training. Especially when some unexpected software behaviors



(a) Traditional Software (b) Intelligent Software

Figure 1.2: Comparison between (a) traditional software (white-box) and (b) intelligent software (black-box) in terms of internal mechanisms.

occur in safety-critical applications such as auto-driving cars and medical diagnosis, intelligent software would be a disaster. To conclude, vulnerability and sensitivity indicate that deep learning models only learn superficial representations and are not reliable.

However, it is unclear why intelligent software makes the wrong prediction on these examples, which impedes the achievement of its reliability. As shown in Figure 1.2, traditional software is a white box with an apparent program logic flow and can be easily interpreted by humans. On the contrary, intelligent software is a black box. It consists of many stacked layers, and each layer is nothing but the combination of linear and non-linear transformation functions with real-value parameters. The internal working mechanisms of intelligent software then is a mystery for humans. Therefore, in the long journey of intelligent software reliability, the interpretability is an essential early step. Intuitively, intelligent software is more reliable if it is more interpretable.

In addition to providing explanations to intelligent software

predictions, interpretability can also help other related tasks, such as testing, debugging, safety. In testing, interpretations can be utilized to design corner or edge cases by covering essential features. In debugging, in analogy to the white-box testing in traditional software, intelligent software can only be debugged when interpretable. An interpretation of a wrong prediction can help understand the cause of the error and derives a direction for bug fixing. As for safety, the interpretation can identify the most critical features, and we may design methods to protect these important features from being attacked.

However, interpreting the intelligent software is non-trivial. Traditional methods heavily rely on the model structure by visualizing the internal components (e.g., attention) as the salience map. However, recent work shows that the attention does not provide meaningful explanations since the relation between attention scores and the model output is unclear [78]. Besides, there is no previous research studying and quantifying the knowledge embedded in intelligent software. In this thesis, we take the state-of-the-art neural machine translation (NMT) models as our intelligent software. The NMT has demonstrated its superiority in real-world deployment, but its interpretability is not thoroughly studied. Specifically, we consider the interpretability of intelligent software from two complementary angles: model attribution and model knowledge:

Model attribution estimation: The task attempts to explain the model input-output behaviors, i.e., the input-output correspondence. In detail, it estimates the importance of each input feature when the model makes a prediction. The obtained interpretation is input-dependent, which provides a local explanation for individual inputs.

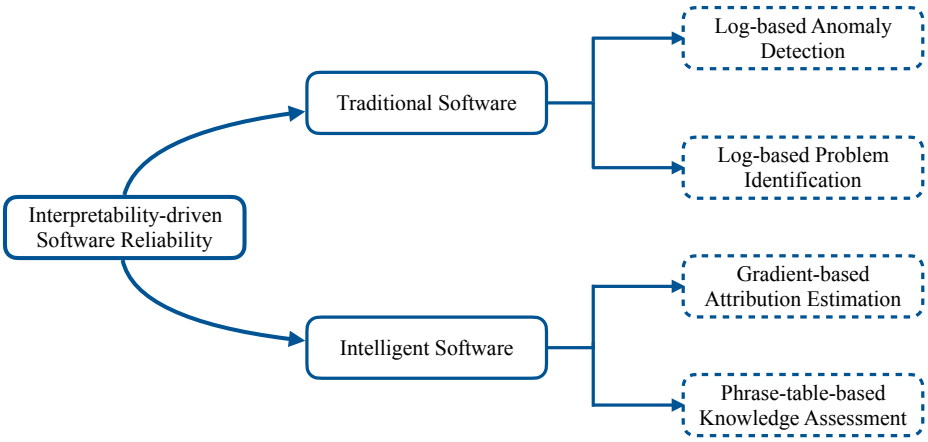


Figure 1.3: Overview of the research in this thesis.

Model knowledge assessment: It focuses on explaining the model behaviors by grounding the uninterpretable model parameters into the human-understandable domain knowledge. The interpretation provides a global explanation for the model and quantitatively assess the knowledge embedded in the model. It also provides a local explanation for individual inputs.

Therefore, the research of this thesis comprises two parts, as depicted in Figure 1.3. In the first part, to realize the traditional software reliability engineering, we interpret the traditional software by applying intelligent methods on the generated logs. This part consists of two closely-related tasks, anomaly detection, and problem identification. In the second part, we focus on intelligent software interpretation for the goal of intelligent software reliability engineering. We first approach the model attribution by providing a gradient-based method, and then quantitatively assess the bilingual knowledge embedded in the intelligent software.

1.2 Thesis Contributions

In this thesis, we mainly focus on software reliability engineering from the interpretability perspective in both traditional software and intelligent software. Studying the interpretability is a crucial early step in realizing the software reliability. Specifically, in traditional software, we approach the software reliability by employing intelligent log-based methods in the anomaly detection and problem identification task. In intelligent software, we propose to attribute the model predictions to the model input with the gradient information. Besides, we explore the method to assess the knowledge embedded in the intelligent software quantitatively. The contributions are summarized as follows:

- For log-based anomaly detection, we provide the first systematic and comprehensive experience report to fill the gap between the industry and academia. To this end, we evaluate six state-of-the-art anomaly detection methods and compare their accuracy and efficiency on two representative production log datasets. Additionally, we release an open-source toolkit of these anomaly detection methods for easy reuse and further study. The open-source toolkit has now become a standard benchmark in this area.
- For log-based problem identification, we propose a novel framework to identify impactful problems, i.e., Log3C. To tackle the challenges of the high imbalance of log distribution and the lack of labeled data, we propose the cascading clustering, an efficient and effective clustering algorithm, and KPI correlation. The proposed Log3C is

evaluated on three real-world log data and applied to the maintenance of actual online service systems.

- For gradient-based attribution estimation, we employ the integrated gradients method to calculate the word importance, which explains the neural machine translation model prediction. We validate that the gradient information can outperform existing interpretation methods such as attention in various language pairs, directions, and model structures. Further analyses of linguistic properties provide some guidance on future model structure design. Besides, we apply our method to detect the under-translation error.

- For phrase-table-based knowledge assessment, we approach the model interpretability by an original bilingual knowledge assessing method in neural machine translation. We leverage the phrase table, an interpretable table of bilingual lexicons, to represent the model knowledge. Massive experiments show that the phrase table is reasonable and consistent. Equipped with the phrase table, we obtain some interesting findings in model learning dynamics and model improvement methods. This work opens up a new angle to interpret NMT with statistic models and provides empirical supports for recent advances in improving NMT models.

1.3 Thesis Organization

The remainder of this thesis is organized as follows.

- **Chapter 2**
  In this chapter, we provide a systematic review of the background knowledge and related work. Firstly, we briefly

introduce different methods of interpreting traditional software in §2.1. Then, §2.2 illustrates the general framework of log analysis using intelligent methods for software reliability engineering. §2.3 provides the basic information about the intelligent software we study in this thesis, i.e., neural machine translation, including the task description and standard model structures. At last, §2.4 reviews recent advances in interpreting intelligent software.

- **Chapter 3**
  This chapter presents our experience report on log-based anomaly detection methods. We first introduce the motivation in §3.1 and provide a detailed review of six state-of-the-art log-based anomaly detection methods in §3.2. In §3.3, we empirically evaluate these methods on two log datasets and introduce several interesting findings based on the experimental results. We further provide some discussions in the limitation and the potential directions in §3.4 and conclude the work in §3.5.

- **Chapter 4**
  In this chapter, we introduce a novel log-based method to identify impactful service system problems. We first introduce the background knowledge in §4.1 and then introduce our proposed Log3C framework as well as the details in §4.2. §4.3 presents the experimental results on three real-world datasets, demonstrating that our method is both efficient and effective. §4.4 presents our result discussions, lists the threats to validity, and shares the success story and lessons learned in the real-world deployment. At last, we summarize the work in §4.5.

- **Chapter 5**
  This chapter presents the interpretability study in neural machine translation model, the first step towards its reliability engineering. §5.1 introduces the background and motivation, and §5.2 illustrates our gradient-based method. Through a variety of experiments, we confirm the consistent effectiveness of the proposed method in §5.3. We provide some linguistic analysis and the application of under-translation error detection in §5.4. Besides, we discuss some possible directions for future explore in §5.5 and conclude the work in §5.6.

- **Chapter 6**
  In this chapter, we propose interpreting the neural machine translation model by assessing its learned bilingual knowledge with statistic models – phrase table. We first introduce the evolvement of the translation knowledge throughout different machine translation generations in §6.1. Next, our proposed phrase table method is illustrated in §6.2 and evaluated under different configurations in §6.3. Equipped with the interpretable phrase table, we analyze the NMT model learning and advanced techniques in §6.4 and summarize the work in §6.6.

- **Chapter 7**
  The last chapter first summarizes this thesis in §7.1. Then in §7.2, we discuss some potential future research directions about software reliability engineering in terms of both traditional software and intelligent software.

□ **End of chapter.**

Chapter 2

Background Review

This chapter reviews the background knowledge and related work. The overall structure is illustrated in Figure 2.1. We first present different types of traditional software interpretation methods in Section 2.1, among which the log analysis is a crucial thread of research. In Section 2.2, we illustrate the general framework for log-based reliability engineering. After that, Section 2.3 introduces the necessary information (e.g., task definition, model structure) about the intelligent software we study in this thesis, i.e., neural machine translation. Then, in Section 2.4, we provide a taxonomy on recent studies in interpreting intelligent software.

2.1 Interpretation for Traditional Software

Although white-box, traditional software such as distributed systems and large-scale office software are hard to comprehend since they usually consist of millions of code lines, which further poses challenges to software reliability engineering. In the last decades, many software engineering practices and program analysis methods have been developed to relieve the problem of

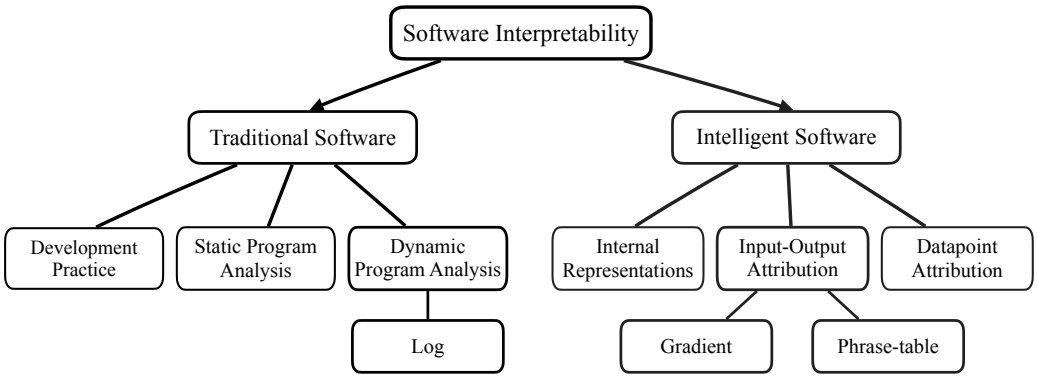


Figure 2.1: An overview of software interpretability research.

software interpretability. In general, there are three categories of methods in interpreting traditional software behaviors: development practices (§2.1.1), static program analysis (§2.1.2) and dynamic program analysis (§2.1.3). Note that we do not aim to explore the entire space but introduce techniques closely related to software reliability engineering.

2.1.1 Development Practices

A natural choice to improve the software interpretability is increasing the readability of source code. Unreadable code is prone to cause bugs in operation. In the early years, studies [47] have found that improving the source code readability can drastically reduce the time to understand it and help the further modification. Industrial practices [135] also demonstrate that software readability is one way to determine whether the software product is reliable.

There are multiple methods to increase the program readability, such as writing code comments, and following naming conventions for objects (e.g., variables, classes). A comment in the source code is a human-readable explanation or annotation of

a code block, function, module, and program. The comments are generally ignored by compilers and interpreters, and will not affect the program behaviors. With the interpretable comments available, developers and maintainers do not need to read the complex code to understand its functionality.
However, the method is labor-intensive and inefficient. With the development of software engineering, more advanced techniques are proposed to interpret and analyze the software behaviors regarding properties such as robustness, correctness, and safety. In the following sections, we briefly introduce the static program analysis and dynamic program analysis.

2.1.2 Static Program Analysis

In static program analysis, the program is not executed, and all analyses are built upon the source code only. Typical techniques that could interpret program behaviors consist of control-flow analysis, data-flow analysis, abstract interpretation, etc. We will briefly explain the key ideas behind these techniques:

- *control-flow analysis*: It extracts information about the function dependency in the program, i.e., which function can be called during the program execution. The dependency information is illustrated by a control flow graph (CFG).

- *data-flow analysis*: Similar to the control-flow analysis, data-flow analysis depicts the variable values at each program point and their changes over time. The information is represented by the data-flow diagram (DFD).

- *abstract interpretation*: It extracts the information about possible executions of a program by the form of mathemat-

ical characterizations; in other words, semantics.

Although interpreting the software behaviors from different perspectives, the techniques mentioned above provide necessary information for locating potentially vulnerable code, especially in safety-critical software. Since the static program analysis does not execute the software, it might lead to inaccurate results. On the contrary, dynamic program analysis resolves the problem by running the source code.

2.1.3 Dynamic Program Analysis

Dynamic analysis is proposed to leverage the runtime knowledge to interpret the software program to increase the analysis accuracy. Three typical analysis methods are testing, monitoring, and program slicing. The details are presented as follows:

- *testing*: As a standard procedure in software development, the software is tested with tailored test cases to ensure its normal operations. Testing methods interpret the software by ensuring the program behaviors consistent with the expected output when feeding an input.

- *monitoring*: Monitoring can effectively interpret the software execution by inserting numerous monitoring indicators into the source code. During the runtime, the software collects information about the program, such as resource usage (e.g., CPU utilization rate, memory consumption) and recording events (such as logs). Particularly in software maintenance, log plays a vital role in monitoring software execution and interpreting software behaviors.

- *program slicing*: It reduces the program to the minimum form but still produces the selected program behaviors. Program slicing provides a faithful representation of the original program. It could interpret the original program by a simplified but equally-functioned program.

In this thesis, a part of our focus lies in the log-based analysis for software reliability engineering. Unlike the techniques mentioned above, we aim to provide intelligent solutions to analyze the massive log data better. In the following section, we will introduce the general framework for log-based analysis with intelligent methods, i.e., machine learning techniques.

2.2 Intelligent Log Analysis Framework

Due to the great volume of log data, interpreting the traditional software becomes tedious and difficult, if not possible. In recent years, intelligent log analysis has been widely employed to improve the reliability of traditional software systems in many aspects [111], such as the anomaly detection [16, 93, 159], failure diagnosis [32, 100, 117], program verification [21, 130], and performance prediction [28].

In this part, we will illustrate the general framework of log analysis for traditional software reliability engineering. As depicted in Figure 2.2, the log analysis framework consists of four steps: log collection (§2.2.1), log parsing (§2.2.2), feature extraction (§2.2.3) and log mining (§2.2.4). Details of each step are presented in the following sections.

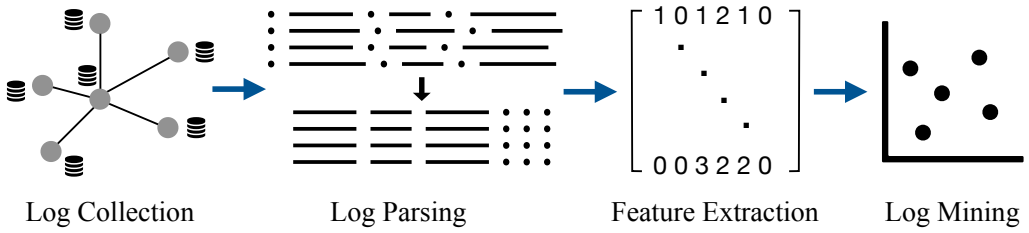


Figure 2.2: A framework of intelligent log analysis.

2.2.1 Log Collection

As aforementioned in Section 1.1 and illustrated in Figure 1.1, logs are firstly generated when executing the logging statements in the software runtime phase and then saved on the local disk. In large-scale traditional software, for example, a distributed system that contains tens of thousands of computing nodes. It routinely generates logs to record system states and runtime information, each comprising a timestamp and a log message indicating when and what has happened. Generally, logs are firstly generated on each node and then gather for the downstream tasks such as manual inspecting and automated analysis. For example, Figure 2.3 shows 8 log messages extracted from a service system that recording detailed information about the end-user HTTP requests.

Due to the enormous volume of logs (e.g., Petabytes per day) that large-scale systems may generate, the storage becomes trouble even with distributed file systems such as HDFS. Recent studies [91, 31] propose to compress the original log file to dramatically reduce the storage space, which we believe is a promising research direction.

2.2.2 Log Parsing

Log parsing aims to extract the log event (i.e., event template) from the raw log messages, as depicted in Figure 2.3. Logs are unstructured plain text that consists of constant parts and variable parts, where the constant parts keep unchanged while the variable parts may vary in different program executions. For instance, in Figure 2.3, log message 2 and 8 share the same logging template "Leaving Monitored Scope (*) Execution Time=*", while the remaining strings in the logs are variable parts as they are not fixed in different messages. Constant parts are predefined in the logging statements by developers. Variable parts are often generated dynamically (e.g., port number, IP address) and often not be well utilized in the downstream tasks. The most straightforward way of log parsing is to write a regular expression for every logging statement in the source code, as adopted in [159]. However, it is tedious and time-consuming because the source code updates frequently and is not always available in practice (e.g., third-party libraries). Thus, automatic log parsing without source code is imperative.
There are mainly three types of automated log parsing methods: clustering (e.g., LKE [53], LogSig [142], LogMine [64]), frequent-pattern mining (SLCT [146], LogCluster [147]) and heuristic-based (e.g., iPLoM [98], Drain [71], AEL [79]).

- *clustering*: In clustering-based log parsers, we calculate the distances between logs first, and then clustering techniques are often employed to separate logs into different groups. Finally, a log event is generated from each cluster.

- *frequent-pattern mining*: For frequent-pattern mining approaches, frequent item-sets (e.g., tokens, token-position

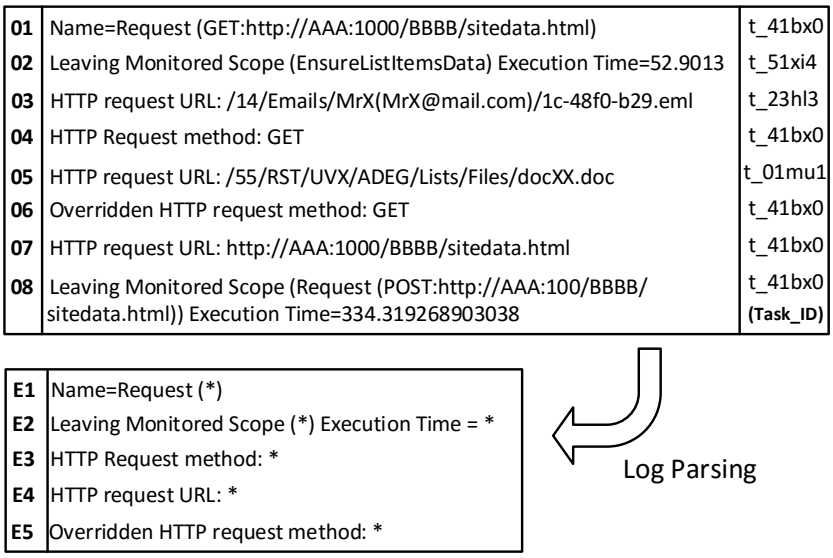


Figure 2.3: An example of log messages and log events.

pairs) are built first by traversing over the log data. Next, frequent words are selected and composed as the event candidates. Finally, candidates are further processed to become log events.

- *heuristic*: Heuristic-based methods parse logs by making use of the characteristics of logs. For example, Drain utilizes a fixed-depth tree structure to represent log messages and extracts log events efficiently.

In our previous study, we evaluate 13 log parsers on a total of 16 log datasets in our previous work [169, 70]. Besides, we published an open-source log parsing toolkit online[1], which is employed to parse raw logs into log events in this thesis.

[1]Log parsers available at https://github.com/logpai/logparser

2.2.3 Feature Extraction

A single log cannot reflect the problem due to its limited information. Instead, a common practice is to leverage a sequence of logs, i.e., log sequence. However, in large-scale systems, different log sequences are often interleaved and cannot be disentangled easily. To form a log sequence, log messages that have strong relations should be linked together, and details are as follows:

- *task identifier*: Task identifier, such as the job id and the process id is to mark different execution flows for a large system or software in runtime. For instance, HDFS logs use the block_id to record the allocation, writing, replication, deletion of a certain block. Thus, Logs that share the same task identifier can then be linked together as a log sequence.
- *time-stamp*: Since not all logs have the preset task identifier, an alternative way is to leverage the timestamp information. Logs that are generated in the same time interval compose a log sequence. There are also two configurations for the time interval setting: fixed window and sliding window, as shown in Figure 2.4. For the fixed window method, logs that occur in the same time interval (i.e., window size such as 10 minutes) are grouped together, and different time intervals have no overlap. Differently, logs in different sliding windows have overlap. The sliding window consists of two attributes: window size and step size, where the window size is the same as the one in the fixed window, and the step size defines the time interval before the next sliding window.

After linking the log sequence, we construct a numerical feature

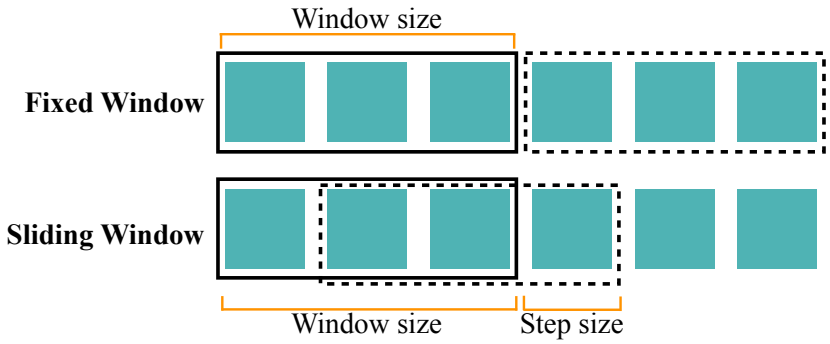


Figure 2.4: An illustration on time window (fixed window and sliding window) for log sequence.

vector. In detail, given the log sequence and the parsing method, we take the log event as a feature and use a feature vector to represent the log sequence. In each log sequence, we count the occurrence number of each log event to form the feature vector (namely, feature vector). For example, if the feature vector is $[0, 0, 2, 3, 0, 1, 0]$ and each position represent a log event, it means that event 3 occurred twice, and event 4 occurred three times in this log sequence. In this way, we can successfully reconstruct the original log sequence and represent it using the feature vector.

2.2.4 Log Mining

Based on the extracted feature vector, we aim to mine some useful patterns with intelligent methods in this step. Since this part is also our main contributions in this thesis, we will briefly introduce some basic ideas and leave more detailed descriptions in Chapter 3 and Chapter 4.

The general flow for log mining is first to train the model based on the training data and then test on a reserved subset of test data to evaluate its performance. Each data sample is denoted

as (x, y), where x is the feature vector of a log sequence, and y is the data label (might not be available), such as the anomaly or not. In terms of the existence of data label y, the log mining methods can be divided into two threads: *supervised method* and *unsupervised method*. In the supervised method, labels are used to guide the training of the supervised models. Typical supervised methods include the decision tree, logistic regression, neural networks. While in unsupervised methods, there is no label for the model training. Typical methods are the dimension reduction method, such as PCA, one-class SVM, and clustering. There are a variety of log analysis tasks, as aforementioned, in this thesis, we mainly focus on two tasks: anomaly detection and problem identification. We review some related researches as follows:

Anomaly detection task aims at finding abnormal behaviors, which can be reported to the developers for manual inspection and debugging. Xu et al. [159] propose the first anomaly detection method based on PCA in the HDFS system and Beschastnikh et al. [21] formalize the logs as a finite state machine to describes system runtime behaviors and detect anomalies. Besides, Farshchi et al. [49] adopt a regression-based analysis technique to detect anomalies of cloud application operations. There are also other studies [23, 150, 12, 7, 11] working on the anomaly detection in different software systems for different purposes. Different from these methods that focus on detecting a specific kind of anomaly, in Chapter 3, we evaluate the effectiveness and efficiency of various anomaly detection methods for anomalies in large scale distributed systems.

Problem identification task is an extension to the anomaly detection, which categorizes different problem types by grouping

similar log problems together. Lin et al. [90] proposed a clustering-based approach for problem identification. Based on testing environment logs, a knowledge base is built firstly and updated in the production environment. However, it requires manual examination when new problems appear. Yuan et al. [162] employed a classification method to categorize system traces by calculating the similarity with traces of existing and known problems. Beschastnikh et al. [20] inferred system behaviors by utilizing logs, which can support anomaly detection and bug finding. Ding et al. [40, 41] correlated logs with system problems and mitigation solutions when similar logs appear. Shang et al. [130] identified problems by grouping the same log sequences after removing repetition and permutations. However, they ignored the different importance of log events and the similarity between any two log sequences. Chapter 4 presents our attempts on log-based problem identification, in which we propose an efficient and effective clustering algorithm under the guidance of system performance monitoring.

2.3 Intelligent Software

In this section, we present the details of intelligent software that we studied in this thesis, i.e., Neural Machine Translation (NMT). We start with the machine translation task and then introduce the RNNSearch model and the state-of-the-art Transformer model for NMT.

2.3.1 Neural Machine Translation Task

Machine translation is the task that aims to convert an input sentence in the source language into an output sentence in

the target language without losing any information. Machine translation has now been widely deployed in many industrial products, e.g., Google Translate, social network posts translation.

Suppose I is the source sentence length and J is the length of the target sentence, the goal of a machine translation model M: $\mathbf{x} \rightarrow \mathbf{y}$ is to maximize the conditional probability of the target sequence $\mathbf{y} = \{y_1, ..., y_J\}$ given a source sentence $\mathbf{x} = \{x_1, ..., x_I\}$, :

$$P(\mathbf{y}|\mathbf{x}; \boldsymbol{\theta}) = \prod_{j=1}^{J} P(y_j|\mathbf{y}_{<j}, \mathbf{x}; \boldsymbol{\theta})$$

where $\boldsymbol{\theta}$ is the model parameter and $\mathbf{y}_{<j}$ is a partial translation. At each time step j, the model generates an output word of the highest probability based on the source sentence $\mathbf{x}$ and the partial translation $\mathbf{y}_{<j}$.

Training: The model is trained on N parallel samples (x, y) (training corpus D), where N is usually very large, such as around 4.5 million in WMT English⇒German translation corpus. The training objective is to minimize the negative log-likelihood loss on the training corpus:

$$\mathcal{L}(\boldsymbol{\theta}; D) = -\sum_{(\mathbf{x},\mathbf{y})\in D}^{N} \log P(\mathbf{y}|\mathbf{x})$$

Inference: After training the model, we can employ the model to decode the output sentence given any input sentence in the source language. Since the greedy decoding may generate less satisfactory sentences, during the inference, beam search, is widely adopted to decode a more optimal translation. Beam search is now the standard practice for machine translation tasks, generating tokens with top-beam (e.g., top 5) highest probability.

Evaluation: BLEU [115] is the standard metric for evaluating the machine translation performance. It measures the similarity between the generated translation and the reference sentence from the token matching perspective, i.e., 1-gram to 4-gram. We usually use the 4-gram NIST BLEU score as the evaluation metric.

2.3.2 Model Structures

The machine translation task has a long history, in which the MT model have evolved from MT (RBMT) [67, 126], through SMT [25, 110], to NMT [141, 15]. RBMT methods require large sets of linguistic rules and extensive lexicons with morphological, syntactic, and semantic information, which are manually constructed by humans. Benefiting from the availability of large amounts of parallel data in the 1990s, SMT approaches relieve the labor-intensive problem of RBMT by automatically learning the linguistic knowledge from bilingual corpora with statistic models. More recently, NMT, which builds a single end-to-end neural network on the training corpora, has taken MT's field.
Both the RBMT and SMT models are interpretable to humans since they involve many manually designed features that carry real-world meanings, e.g., word alignment. However, NMT models, which represent the current state-of-the-art techniques in MT task, are uninterpretable, which hinders the achieving of its reliability. Therefore, we will introduce our recent work on NMT model interpretability in this thesis. We will briefly go through two representative models in NMT in the following part:

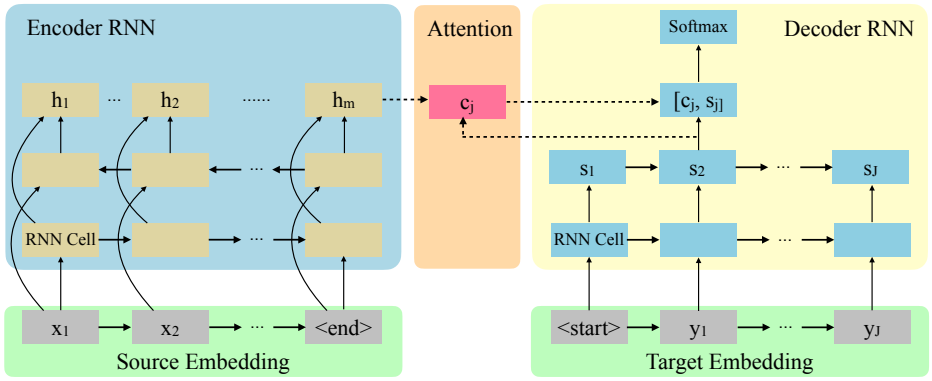


Figure 2.5: RNNSearch structure in neural machine translation.

RNNSearch model As shown in Figure 2.5, the RNNSearch [15] model is the first model that defines the Encoder-Decoder model structure and dramatically boosts the neural machine translation performance. The RNNSearch model relies mostly on the recurrent neural network (RNN), which is good at modeling sequential data such as the natural language.

In detail, the RNNSearch model consists of three components: encoder, decoder as well as the attention module. The encoder is a bi-directional RNN built on the source embedding of the input sentence ($[x_1, x_2, ..., end]$), and it can capture the information from both forward and backward direction. The decoder is a standard forward RNN that predicts the next token based on previously generated tokens. Besides, according to the hidden state in the current decoder step, the RNNSearch model utilizes the attention mechanism to treat the encoder hidden states (from h_1 to h_m) differently. The attention mechanism firstly calculates the attention weight (a_{ji}) of different encoder hidden states and takes the weighted sum of encoder hidden states as the context vector:

$$c_j = \sum_{i=1}^{m} a_{ji} h_i$$

In the decoder generation, the context vector considers the input tokens' contribution and assigns more weights to more relevant tokens. Therefore, attention is often used as an interpretation method in natural language processing tasks such as machine translation.

Transformer model The RNNSearch model has two significant drawbacks: 1) It suffers from the long-range dependency problem since the RNN structure tends to forget tokens on distant positions. 2) The primary component RNN in the RNNSearch model has to iterate through the sequence one by one word, and each state depends on all previous states. The sequentiality is an obstacle to the parallelization of model training, especially for modern computing devices that rely on parallel processing, such as Tensor Processing Units (TPUs) and Graphics Processing Units (GPUs).

To tackle the problems as mentioned above, Transformer [149] employs a novel self-attention module to parallelize the calculation and further accelerate the training. Transformer is now the state-of-the-art neural machine translation model and our intelligent software of interest in the following chapters.

Similar to the RNNSearch model, the Transformer model is also composed of the encoder and decoder components. In both the encoder and decoder component, firstly, each input word is represented as the addition of an embedding vector and a positional embedding which captures the relative position within the sequence. The representation is then fed into modules that can be stacked on top of each other multiple times, as depicted by Nx in Figure 2.6.

Each module consists mainly of a multi-head self-attention and a

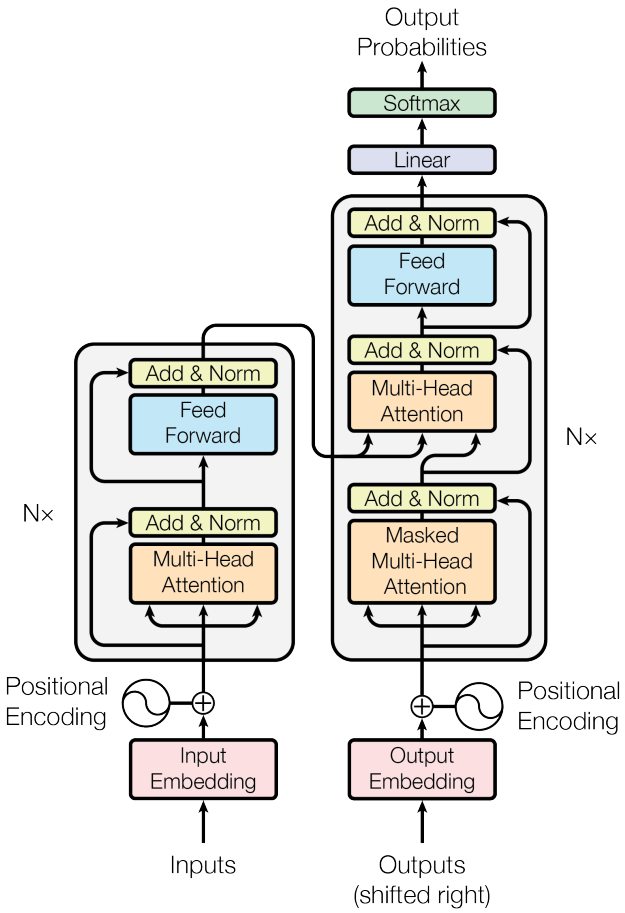


Figure 2.6: Transformer structure in neural machine translation.

feed-forward network. Unlike traditional methods that compute a single attention weight, the multi-head self-attention computes multiple attention blocks using the multi-head, concatenates, and projects them linearly back to a predefined subspace. Each attention head calculates the scaled dot-product attention with different linear projections over the given input representations. Finally, a fully connected feed-forward network is leveraged, which has two linear transformation layers with a ReLU activation function [63].

The decoder differs from the encoder slightly. The decoder generates the word from left to right, and thereby the multi-head self-attention attends only to past words. Besides, there is an encoder-decoder self-attention module which attends to the encoder representations. Finally, a Softmax layer is employed to map the output representation to the probability distribution over the target vocabulary.

In Chapter 5 and Chapter 6, we study the interpretability of NMT models, and our main experimental results are based on the state-of-the-art Transformer model.

2.4 Interpretation for Intelligent Software

In this section, we will first introduce the overall taxonomy for interpreting the intelligent software. Generally, the interpretability is approached from three threads: input-output attribution, internal representations, and data point attribution. Details are illustrated in the following sections.

2.4.1 Input-Output Attribution

To explain why the model makes a correct or incorrect prediction, a straightforward way is to understand the prediction from the input feature perspective [8, 42, 72]. In other words, input-output attribution attempts to estimate the contribution that each input feature makes to the output prediction, e.g., feature importance. Since features in different inputs may play different roles in the model prediction, the attribution method is a local-explanation method which is input-specific. The attribution method contains two major groups, perturbation-based [8] and gradient-based [140, 72].

The perturbation-based methods [123, 170, 50] estimate the input feature contribution by perturbing (or removing) the feature and measuring the performance change of model predictions after this operation. However, perturbation-based methods are easy to implement but often very slow. Besides, the perturbed feature numbers and feature orders can also significantly affect the resulting interpretation, making it difficult to rely on the results. They do not exploit any intermediate information, such as gradients. For example, Alvarez-Melis et al. [8] measure the relevance between two input-output tokens by perturbing the input sequence.

The gradient-based method computes attributions by taking a few backward passes through the network. Specifically, we estimate the attribution values as the gradient of the model prediction with respect to the input features. However, directly applying the gradient could be problematic due to the non-linearity in the deep neural networks. To tackle the problem, recent advances have been proposed such as the Layer-wise Relevance Propagation (LRP) in several variants [13, 103],

DeepLIFT [134, 133], and Integrated Gradients [140, 128, 105, 37]. Among all gradient-based approaches, the integrated gradients [140] is appealing since it does not need any instrumentation of the architecture and can be computed easily by calling gradient operations. In the Chapter 5 of this thesis, we employ the IG method to interpret NMT models and reveal several interesting findings, which can potentially help debug NMT models and design better architectures for specific language pairs.

2.4.2 Internal Representations

The internal representation based methods focus on the model internals instead of the input-output correspondence, which provides a global explanation to human. In general, the goal of internal representation interpretability is to understand the knowledge embedded in the learned representation vectors. There are mainly three threads of studies: layer representation [131, 18, 151], neurons [17], model weights [152, 78, 155].
In layer representation understanding, for example, in the NMT task, Several researchers turn to expose systematic differences between human and NMT translations [88, 127], indicating the linguistic properties worthy of investigating. They mainly focus on whether the NMT model embeds sufficient linguistic knowledge, such as grammar for translation. To achieve so, the probing task [34] is utilized to evaluate the linguistic information embedded in the representation. However, the learned representations may depend on the model implementation, which potentially limits the applicability of these methods to a broader range of model architectures.
Recently, the functionality of specific neurons also attracts the

interest of researchers. In deep learning models, are there some neurons that are more important than other neurons? Does a particular neuron control a property of the generated sentences? For example, researchers [17] show that some specific neurons control the gender property in the generated sentence. These studies attempt to open the black-box of deep learning by probing into the neurons.

The model weights are mostly real-value parameters that cannot be well-explained, but the attention module is often considered as self-explainable. However, recent researches show that attention is not an explanation, and different attention distribution may yield the same model predictions [152, 78, 155].

2.4.3 Datapoint Attribution

Datapoint attribution is a slightly different method that instead focuses on the data samples. In data point attribution, counterfactual explanation and training sample explanation are two standard sets of methods.

In counterfactual explanation, to explain the prediction of a data sample, we need to first find a similar data samples by changing some input features that may change the model prediction in a relevant way, for example, a flip in the predicted class.

To explain the prediction of a test sample, the method finds similar data points in the training data relevant to the current data sample prediction, which may further help identify the prediction problems [95, 139, 86]. For example, the method is useful when finding out that the training data is poisoned. Removing the problematic training samples could then help avoid the wrong prediction problem.

The data point attribution provides another angel for interpret-

ing the model prediction, which is also relevant to our studies on the bilingual knowledge assessment since the extraction of bilingual knowledge is also training data based.

□ **End of chapter.**

Chapter 3

Log-based Interpretation for Anomaly Detection

In the reliability engineering of traditional software such as large-scale distributed systems, anomaly detection plays an important role. Logs which record system runtime information, are widely used to interpret the software behaviors in terms of the anomaly detection. In this chapter, we mainly focus on system anomaly detection. Specifically, we provide an empirical study on existing anomaly detection methods and evaluate their performance regarding effectiveness and efficiency on two representative log datasets. We further conclude several interesting findings that might guide future research and release the toolkit for public reuse. The chapter is organized as follows: we first introduce the problem background in §3.1 and present a detail review on existing methods in §3.2. We then show the experiments and the findings in §3.3. We discuss the limitations and the potential directions in §3.4 and conclude this chapter in §3.5.

3.1 Problems and Motivation

Modern software systems are evolving to a large scale, either by scaling out to distributed systems built on thousands of commodity machines (e.g., Hadoop [1], Spark [2]) or by scaling up to high-performance computing with supercomputers of thousands of processors (e.g., Blue Gene/L [112]). These systems are emerging as the core part of the IT industry, supporting a wide variety of online services and intelligent applications for millions of users. Because most of these systems often operate on a 24x7 basis, serving millions of online users globally, high availability and reliability become a must. Any incidents of these systems, including service outage and degradation of service quality, will break down applications and lead to significant revenue loss.

Anomaly detection, which aims at uncovering abnormal system behaviors promptly, plays a vital role in incident management of large-scale systems. Timely anomaly detection allows system developers (or operators) to pinpoint issues promptly and resolve them immediately, thereby reducing system downtime. Systems routinely generate logs, which record detailed runtime information during system operation. Such widely-available logs are used as the primary data source for system anomaly detection. Log-based anomaly detection (e.g., [90, 117, 159]) has become a research topic of practical importance both in academia and in industry. However, traditional manual inspecting methods become not applicable due to the reasons we introduced in Section 1.1: 1) the vast volume of logs makes it difficult to inspect the logs manually; 2) the large scale systems are too complex to understand by a single developer; 3) the fault tolerance mechanism may confuse the developers and lead

to many false positives. As a result, automated log analysis methods for anomaly detection are highly in demand.

Log-based anomaly detection has been widely studied in the last decades. However, we found that there is a gap between research in academia and practice in the industry. On the one hand, developers are, in many cases, not aware of the state-of-the-art anomaly detection methods, since there is currently a lack of a comprehensive review on this subject. They have to go through a large body of literature to get a comprehensive view of current anomaly detection methods. The literature review is a cumbersome task yet does not guarantee that the most suitable method can be found since each research work usually focuses specifically on reporting a particular method towards a target system. The difficulty may be exacerbated if developers have no prior background knowledge of machine learning required to understand these methods.

On the other hand, to our knowledge, no log-based open-source tools are currently available for anomaly detection. There is also a lack of comparison among existing anomaly detection methods. It is hard for developers to know which is the best method for their practical problems at hand. To compare all candidate methods, they need to try each one with their implementation. Enormous efforts are often required to reproduce the methods because no test oracles exist to guarantee correct implementations of the underlying machine learning algorithms.

To bridge this gap, in this chapter, we provide a detailed review and evaluation of log-based anomaly detection, as well as release an open-source toolkit[1] for anomaly detection. Our goal is not to improve any specific method, but to portray an overall picture

[1] Available at https://github.com/logpai/loglizer

of current research on log analysis for anomaly detection. We believe that our work can benefit researchers and practitioners in two aspects: Firstly, the review can help them grasp a quick understanding of current anomaly detection methods. Secondly, the open-source toolkit avoids time-consuming yet redundant efforts for re-implementation, which allows them to reuse existing methods and make further customization or improvement easily.

The log analysis process for anomaly detection involves four main steps: log collection, log parsing, feature extraction, and anomaly detection, as depicted in 2.2. In this chapter, we will focus primarily on the aspects of feature extraction and machine learning models for anomaly detection. According to the type of data involved and the machine learning techniques employed, anomaly detection methods contain two main categories: supervised anomaly detection and unsupervised anomaly detection. Supervised methods require the training labels with clear specifications on normal instances and abnormal instances. Then classification techniques are utilized to learn a model to maximize the discrimination between normal and abnormal instances. Unsupervised methods, however, do not need labels at all. They work based on the observation that an abnormal instance usually manifests as an outlier point distant from other instances. As such, unsupervised learning techniques, such as clustering, can be applied. More specifically, we reviewed and implemented six representative anomaly detection methods reported in recent literature, including three supervised methods (i.e., Logistic Regression [22], Decision Tree [27], and SVM [89]) and three unsupervised methods (i.e., Log Clustering [90], PCA [159], and Invariant Mining [93]). We further perform

a systematic evaluation of these methods on two publicly-available log datasets, with a total of 15,923,592 log messages and 365,298 anomaly instances. The evaluation results are reported on *precision*, *recall*, *F-measure* and *efficiency*. Though the data are limited, we believe that these results, as well as the corresponding findings revealed, can provide guidelines for the adoption of these methods and serve as baselines in future development.

3.2 Methodology

As aforementioned in Section 2.2, anomaly detection is a specific log mining task. Before the logging mining phase, we have the log collection, log parsing, feature extraction phases, which have been introduced in detail. The input to anomaly detection is the feature vector for each log sequence, while the output is whether the log sequence is an anomaly or not. In this section, we mainly focus on providing a detailed review of six representative anomaly existing detection methods. Among them, three are supervised while the other three are unsupervised. The supervised methods require the training data to have labels while the unsupervised methods do not have the label information. We present the details as follows.

3.2.1 Supervised Anomaly Detection

Supervised learning (e.g., decision tree) is a kind of machine learning task which derives the model from labeled training data. Labeled training data, which indicate normal or abnormal state by labels, are the prerequisite of supervised anomaly detection. The more labeled training data we have, the more precise

the model would be. We will introduce three representative supervised methods: Logistic regression, Decision tree, and Support vector machine (SVM) in the following.

1) Logistic Regression

Logistic regression is a widely used statistical model for classification. To decide whether an instance is normal or abnormal, we use logistic regression to estimate the probability p of all possible states. The probability p is estimated by a logistic function that built on labeled training data. When a new instance appears, the logistic function could compute the probability p $(0 < p < 1)$ of all possible states. After obtaining the probabilities, the state of the largest probability is the classification output.
To detect anomalies, we construct a feature vector from each log sequence, and every feature vector, as well as with its label, is called an instance. Firstly, we use training instances to establish the logistic regression model, which is a logistic function. After obtaining the model, we feed a testing instance X into the logistic function to compute its possibility p as an anomaly, the label of X is anomalous when $p \geq 0.5$ and normal otherwise.

2) Decision Tree

Decision Tree is a tree structure diagram that uses branches to illustrate the predicted state for each instance. The decision tree is constructed in a top-down manner using the training data. Each tree node is created using the current "best" attribute selected by attribute's information gain [65]. For example, the root node in Figure 3.1 shows that there are 20 instances in our dataset. When splitting the root node, the occurrence number

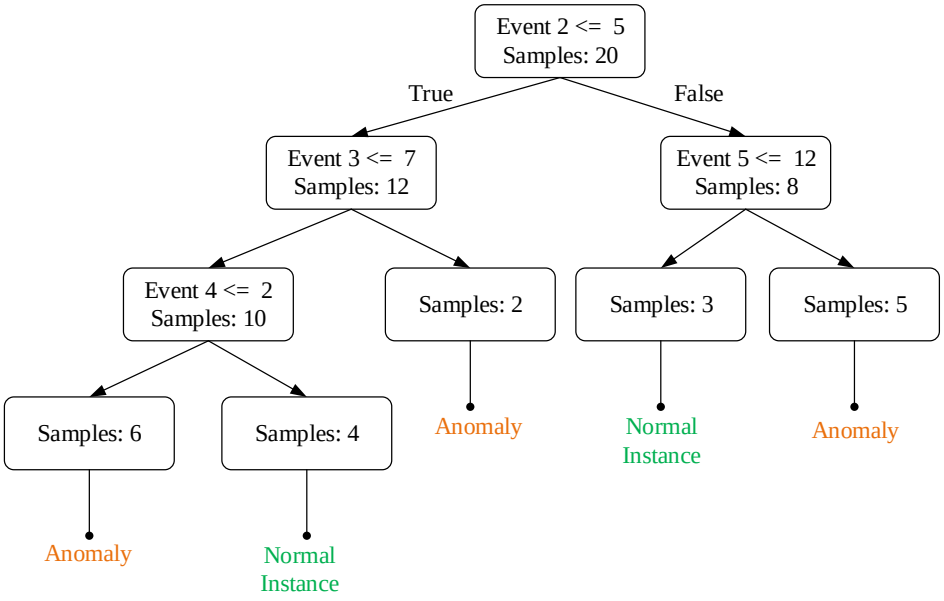


Figure 3.1: An example of anomaly detection based on decision tree.

of Event 2 is selected as the "best" attribute. Thus, the entire 20 training instances are divided into two subsets according to the value of this attribute, in which one contains 12 instances and eight instances, respectively.

Decision Tree was first applied to failure diagnosis for the web request log system in [27]. The feature vectors, together with their labels, are utilized to build the decision tree. To identify the state of a new instance, the method traverses the decision tree according to each traversed tree node's predicates. In the end, the instance will arrive at one of the leaves, which indicates the instance state.

3) SVM

Support Vector Machine (SVM) is a supervised learning method for classification. In SVM, a hyperplane is constructed to

separate various classes of instances in high-dimension space. Finding the hyperplane is an optimization problem, which targets maximizing the distance between the hyperplane and the nearest data point in different classes.
In [89], Liang et al. employ the SVM to detect failures and compare the performance against other methods. Similar to Logistic Regression and Decision Tree, the training instances are feature vectors together with their labels. In anomaly detection via SVM, if a new log sequence locates on above the hyperplane, it would be reported as an anomaly, while marked as normal otherwise. There are two kinds of SVM, namely linear SVM and non-linear SVM. In this chapter, we only discuss linear SVM, because linear SVM outperforms non-linear SVM in most of our experiments.

3.2.2 Unsupervised Anomaly Detection

Unlike supervised methods, unsupervised learning is another common machine learning task, but its training data is unlabeled. Unsupervised methods are more applicable in a real-world production environment due to the lack of labels. Conventional unsupervised approaches include various clustering methods, association rule mining, PCA, etc.

1) Log Clustering

In [90], Lin et al. design a clustering-based method called LogCluster to identify online system problems. LogCluster requires two training phases, namely knowledge base initialization phase, and online learning phase. Thus, the training instances are divided into two parts for these two phases, respectively.

The knowledge base initialization phase contains three steps: log vectorization, log clustering, representative vectors extraction. Firstly, log sequences are vectorized as feature vectors, as previously introduced. Besides, the feature vectors are further revised by Inverse Document Frequency (IDF) [125] and normalization. Secondly, LogCluster clusters normal and abnormal feature vectors separately with agglomerative hierarchical clustering, yielding two sets of vector clusters (i.e., normal clusters and abnormal clusters) as a knowledge base. Finally, we select a representative vector for each cluster by computing its centroid. The online learning phase is used to adjust the clusters constructed in the knowledge base initialization phase. In the online learning phase, feature vectors are added to the knowledge base one by one. Given a feature vector, the distances with existing representative vectors are computed separately. If the smallest distance is less than a threshold, this feature vector will be added to the nearest cluster, and the representative vector of this cluster will be updated. Otherwise, LogCluster creates a new group using this feature vector.

After constructing the knowledge base and complete the online learning process, LogCluster can be employed to detect anomalies. Specifically, to determine the state of a new log sequence, we compute its distance to representative vectors in the knowledge base. If the smallest distance is larger than a threshold, the log sequence is reported as an anomaly. Otherwise, the log sequence is reported as the nearest cluster's status, i.e., normal/abnormal case.

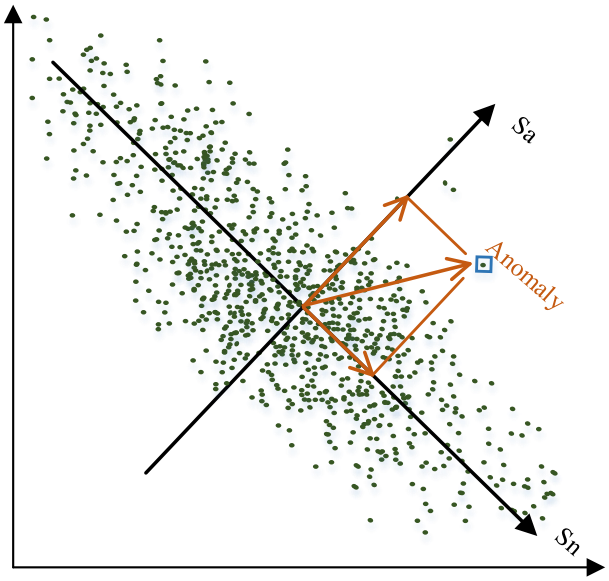


Figure 3.2: An illustration of anomaly detection based on PCA.

2) PCA

Principal Component Analysis (PCA) is a statistical method that has been widely used to conduct dimension reduction. The basic idea behind PCA is to project high-dimension data (e.g., high-dimension points) to a new coordinate system composed of k principal components (i.e., k dimensions), where k is set to be less than the original dimension. PCA calculates the k principal components by finding components (i.e., axes) which catch the most variance among the high-dimension data. Thus, the PCA-transformed low-dimension data can preserve the major characteristics (e.g., the similarity between two points) of the original high-dimension data. For example, in Figure 3.2, PCA attempts to transform two-dimension points to one-dimension points. S_n is selected as the principal component because the distance between points can be best described by mapping them to S_n.

PCA was first applied in log-based anomaly detection by Xu et al. [159]. In their anomaly detection method, each log sequence is vectorized as a feature vector. After that, we employ the PCA to find patterns between the dimensions of feature vectors. Employing PCA, two subspaces are generated, namely the normal space S_n and anomaly space S_a. S_n is constructed by the first k principal components and S_n is constructed by the remaining $(n-k)$, where n is the original dimension. Then, the projection $y_a = (1 - PP^T)y$ of an feature vector y to S_a is calculated, where $P = [v_1, v_2, \ldots, v_k,]$ is the first k principal components. If the distance of y_a is larger than a threshold, the corresponding feature vector will be reported as an anomaly. For example, the selected point in Figure 3.2 is an anomaly because the distance of its projection on S_a is too large. To be specific, a feature vector is regarded as an anomaly if

$$SPE \equiv \|y_a\|^2 > Q_\alpha$$

where squared prediction error (i.e., SPE) represents the "distance", and Q_α is the threshold providing $(1-\alpha)$ confidence level. We follow the original paper to set $Q = 0.001$. For k, we calculate it automatically by adjusting the PCA to capture 95% variance of the data, which is the same as the original paper.

3) Invariants Mining

Program Invariants are the linear relationships that always hold during system running, even with various inputs and under different workloads. Invariants mining was first applied to log-based anomaly detection in [93]. Logs with the same task identifier (e.g., job id, block id in HDFS) often represent the

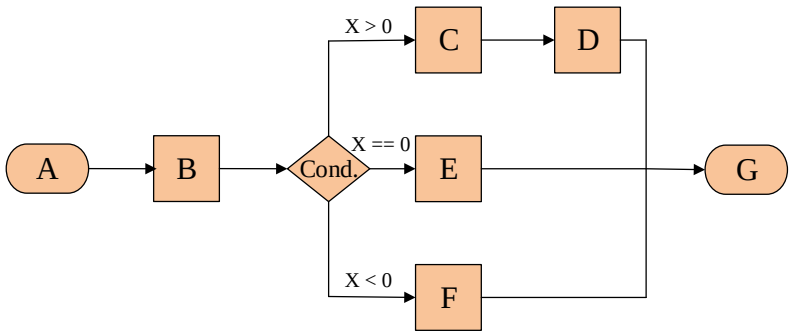


Figure 3.3: An example of the execution flow for a code snippet.

program execution flow of a user request or an operation. A simplified program execution flow is illustrated in Figure 3.3. In this execution flow, the system generates a log message at each stage from A to G. Assuming that there are plenty of instances running in the system and they follow the program execution flow in Figure 3.3, the following equations would be valid:

$n(A) = n(B)$

$n(B) = n(C) + n(E) + n(F)$

$n(C) = n(D)$

$n(G) = n(D) + n(E) + n(F)$

where $n(*)$ represents the number of logs which belong to corresponding log event $*$.

Intuitively, Invariants mining could uncover the linear relationships, e.g., $n(A) = n(B)$, between multiple log events representing the system normal execution behaviors. Linear relationships prevail in real-world system events. For example, normally, a file must be closed after it was opened. Thus, log with the phrase “open file” and log with the phrase “close file” would appear in pairs. If the number of log events “open file” and that of “close file” in an instance are not equal, it will be

marked as abnormal because it violates the linear relationship. Invariants mining, which aims at finding invariants (i.e., linear relationships), contains three steps. The input of invariants mining is a feature vector generated from log sequences, where each row is a feature vector. Firstly, the invariant space is estimated using singular value decomposition, which determines the amount r of invariants that need to be mined in the next step. Secondly, this method finds out the invariants by a brute force search algorithm. Finally, each mined invariant candidate is validated by comparing its support with a threshold (e.g., supported by 98% of the event count vectors). This step will continue until r independent invariants are obtained.
In anomaly detection based on invariants, when a new log sequence arrives, we check whether it obeys the invariants. The log sequence will be reported as an anomaly if at least one invariant is broken.

3.2.3 Comparisons

To reinforce the understanding of the above six anomaly detection approaches and help developers better choose anomaly detection methods to use, we discuss the advantages and disadvantages of different methods in this part.
For supervised methods, labels are required for anomaly detection. The decision tree is more interpretable than the other two methods, as developers can detect anomalies with meaningful explanations (i.e., predicates in tree nodes). Logistic regression cannot solve linearly non-separable problems, which can be solved by SVM using kernels. However, SVM parameters are hard to tune (e.g., penalty parameter), so it often requires much manual effort to establish a model.

Unsupervised methods are more practical and meaningful due to the lack of labels. Log clustering uses the idea of online learning. Therefore, it is suitable for processing a large volume of log data. Invariants mining can not only detect anomalies with high accuracy but can also provide meaningful and intuitive interpretation for each detected anomaly. However, invariants mining is time-consuming. PCA is not easy to understand and is sensitive to the data. Thus, its anomaly detection accuracy varies over different datasets.

3.2.4 Tool Implementation

We implemented six anomaly detection methods in Python with over 4,000 lines of code and packaged them as a toolkit. For supervised methods, we utilize a widely-used machine learning package, scikit-learn [118], to implement the learning models of Logistic Regression, Decision Tree and SVM. There are plenty of parameters in SVM and logistic regression, and we manually tune these parameters to achieve the best results during training. For SVM, we tried different kernels and related parameters one by one, and we found that SVM with linear kernel obtains the better anomaly detection accuracy than other kernels. For logistic regression, different parameters are also explored, and they are carefully tuned to achieve the best performance.
Implementing unsupervised methods, however, is not straightforward. For log clustering, we were not able to directly use the clustering API from scikit-learn since the API is not designed for large-scale datasets, and our data cannot fit the memory. We implemented the clustering algorithm into an online version, whereby each data instance is grouped into a cluster one by one. There are multiple thresholds to be tuned. We also paid great

efforts to implement the invariants mining method, because we built a search space for possible invariants and proposed numerous ways to prune all unnecessary invariants. It is very time-consuming to test different combinations of thresholds. We finally implemented the PCA method according to the original reference based on an API from scikit-learn, which contains two hyper-parameters only and is easy to tune.

3.3 Evaluations

In this section, we will first introduce the datasets and the experiment setup we employed to evaluate these methods. Then, we provide the evaluation results of supervised and unsupervised anomaly detection methods, and these two types of techniques are generally applicable in different settings. Finally, the efficiency of all these methods is evaluated and compared.

3.3.1 Experimental Setup

Log Datasets: Publicly available production logs are scarce data because companies rarely publish them due to confidential issues. Fortunately, by exploring an abundance of literature and intensively contacting the corresponding authors, we have successfully obtained two log datasets, HDFS data [159] and BGL data [112], which are suitable for evaluating existing anomaly detection methods. Both datasets were collected from production systems, with a total of 15,923,592 log messages and 365,298 anomaly samples, manually labeled by the original domain experts. Thus we take these labels (anomaly or not) as the ground truth for accuracy evaluation purposes. Table 3.1 presents more statistical information about the datasets.

System	#Time span	#Data size	#Log messages	#Anomalies
BGL	7 months	708 M	4,747,963	348,460
HDFS	38.7 hours	1.55 G	11,175,629	16,838

Table 3.1: Summary statistics of the datasets.

HDFS data contain 11,175,629 log messages, which were collected from Amazon EC2 platform [159]. HDFS logs record a unique block ID for each block operation such as allocation, writing, replication, deletion. Thus, the operations in logs can be more naturally captured by task ID, as introduced in Section 2.2, because each unique block ID can be utilized to slice the logs into a set of log sequences. Then we extract feature vectors from these log sequences and generate 575,061 feature vectors. Among them, 16,838 samples are marked as anomalies. BGL data contain 4,747,963 log messages, which were recorded by the BlueGene/L supercomputer system at Lawrence Livermore National Labs (LLNL) [112]. Unlike HDFS data, BGL logs have no identifier recorded for each job execution. Thus, we have to use fixed windows or sliding windows to slice logs as log sequences, and then extract the corresponding feature vectors. But the number of windows depends on the chosen window size (and step size). In BGL data, 348,460 log messages have a specific type of failure, and a log sequence is marked as an anomaly if any failure logs exist in that sequence.

Experimental setup: We ran all our experiments on a Linux server with Intel Xeon E5-2670v2 CPU and 128GB DDR3 1600 RAM, on which 64-bit Ubuntu 14.04.2 with Linux kernel 3.16.0 was running. Unless otherwise stated, we repeat each experiment five times and report the average result. We use *precision*, *recall*, and *F-measure*, which are the most com-

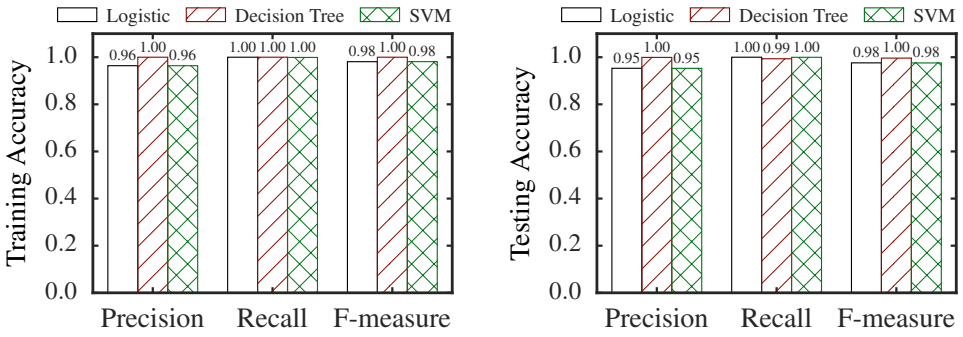
monly used classification metrics, to evaluate the accuracy of anomaly detection methods as we already have the ground truth (anomaly or not) for both of the datasets. As shown in the below equations, precision measures the percentage of how many reported anomalies are correct, recall measures the rate of how many real anomalies are detected, and F-measure indicates the harmonic mean of precision and recall.

$$Precision = \frac{\#Anomalies\ detected}{\#Anomalies\ reported}$$
$$Recall = \frac{\#Anomalies\ detected}{\#All\ anomalies}$$
$$F - measure = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

For all three supervised methods, we choose the first 80% data as the training data, and the remaining 20% as the testing data because only previously happening events could lead to a succeeding anomaly. By default, we set the window size of fixed windows to one hour, and set the window size and step size of sliding windows to be six hours and one hour, respectively.

3.3.2 Effectiveness of Supervised Methods

To explore the effectiveness of supervised methods, we use them to detect anomalies on HDFS data and BGL data. We use task ID to slice HDFS data and then generate the feature vectors, while fixed windows and sliding windows are applied to BGL data separately. To check the validity of three supervised methods (namely Logistic Regression, Decision Tree, SVM), we first train the models on training data and then apply them to testing data. We report both training accuracy and testing accuracy in different settings, as illustrated in the following

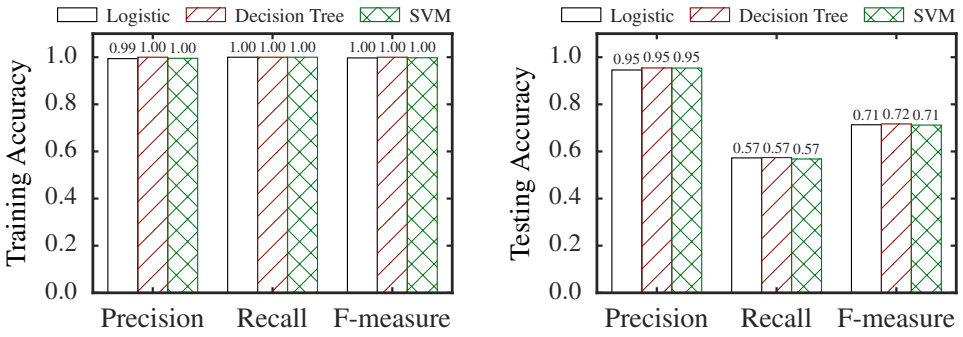


(a) Training Accuracy (b) Testing Accuracy

Figure 3.4: Accuracy of supervised methods on HDFS data with task ID.

Figures. We can observe that all supervised methods achieve high training accuracy (over 0.95), which implies that normal and abnormal instances are well separated by using our feature representation. However, their accuracy on testing data varies with different methods and datasets. The overall accuracy of HDFS data is higher than the accuracy of BGL data with both fixed windows and sliding windows. It is mainly because the HDFS system records relatively simple operations with only 29 log events, which is much less than the 385 log events in BGL data. Besides, HDFS data are grouped by task ID, thereby causing a higher correlation between events in each log sequence. Therefore, anomaly detection methods on HDFS perform better than on BGL.

In particular, Figure 3.4 shows the accuracy of anomaly detection on HDFS data, and all three approaches have excellent performance on testing data with the F-measure close to 1. When applying supervised approaches to the testing data of BGL with fixed windows, they do not achieve high accuracy, although they perform well on training data. As Figure 3.5



(a) Training Accuracy (b) Testing Accuracy

Figure 3.5: Accuracy of supervised methods on BGL data with fixed windows.

illustrates, all three methods on BGL with fixed windows have the recall of only 0.57, while they obtain high detection precision of 0.95. We found that as the fixed window size is only one hour, thus, it may cause the uneven distribution of anomalies. For example, some anomalies that happened in the current window may correlate with events in the former time window, and they are incorrectly divided. Consequently, anomaly detection methods with a one-hour fixed window do not perform well on BGL data.

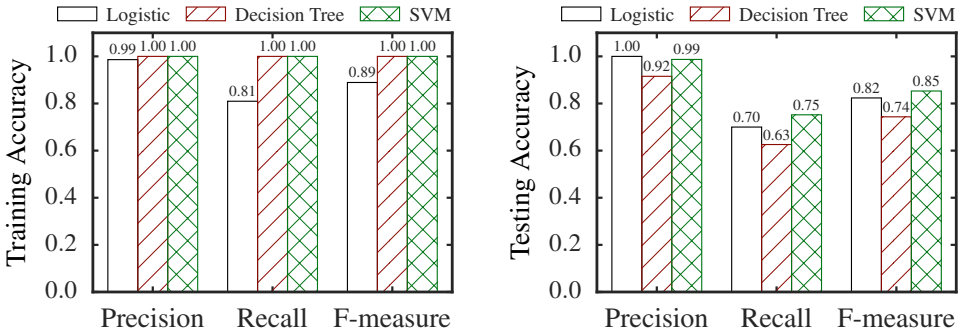
Finding 1: Supervised anomaly detection methods achieve high precision, while the recall varies over different datasets and window settings.

To address the problem of poor performance with fixed windows, we employed the sliding windows to slice BGL data with window size = 6h and step size = 1h. The results are given in Figure 3.6. Comparing with the fixed windows, anomaly detection methods based on sliding windows achieve much higher accuracy on testing data. By using sliding windows, we can not only

Window Size	1 h	3 h	6 h	9 h	12 h
#Sliding windows	5153	5151	5150	5145	5145

Step Size	5 min	30 min	1 h	3 h	6 h
#Sliding windows	61786	10299	5150	1718	860

Table 3.2: Sliding window amount for different window size and step size.

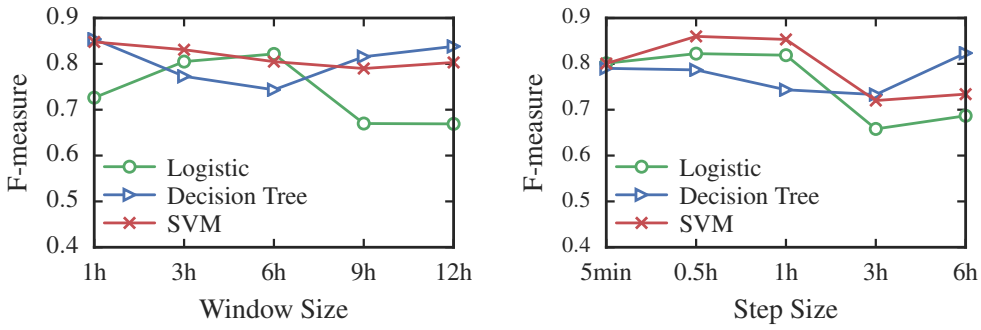


(a) Training Accuracy (b) Testing Accuracy

Figure 3.6: Accuracy of supervised methods on BGL data with sliding windows.

obtain as many windows (feature vectors) as fixed windows but can also avoid the problem of uneven distribution because the window size is much larger. . Among supervised methods, we observe that SVM achieves the best overall accuracy with an F-measure of 0.85. Moreover, the decision tree and logistic regression that are based on sliding windows achieve 10.5% and 31.6% improvements in recall than the results on the fixed windows.

To further study the influences of different window sizes and various step sizes on anomaly detection accuracy, we conduct experiments by changing one parameter while keeping the other parameter constant. According to the diagram a) of Figure 3.7, We hold the step size at one hour while changing the



(a) Different window sizes

(b) Different step sizes

Figure 3.7: Accuracy of supervised methods on BGL data with different window sizes and step sizes.

window size, as shown in Table 3.2. Window sizes larger than 12 hours are not considered as they are not practical in real-world applications. We can observe that the F-measure of SVM slightly decreases when the window size increases. In contrast, the accuracy of logistic regression increases slowly first but falls sharply when window sizes increase to nine hours, and then it keeps steady. Logistic regression achieves the highest accuracy when the window size is 6 hours. The variation trend of decision tree accuracy is opposite to the logistic regression, and it reaches the highest accuracy at 12 hours. Therefore, logistic regression is sensitive to the window size while decision tree and SVM remains stable.

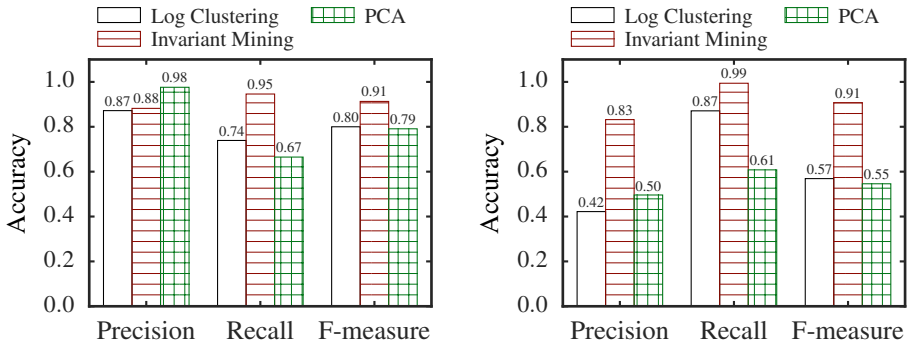
Compared with window size, step size likely has a significant effect on anomaly detection accuracy. Table 3.2 illustrates that if we reduce the step size while keeping the window size at six hours, the number of sliding windows (data instances) increases dramatically. All three methods show the same trend that the accuracy first increases slightly, then drop at around 3 hours. It may be caused by the dramatic decrease in data instance

amount when using a large step size, for example, at 3 hours. One exception is the step size of six hours when the window size equals the step size. Thus, the sliding window is the fixed window. In this situation, some noises caused by overlapping are removed, which leads to a small increase in detection accuracy.

Finding 2: Anomaly detection with sliding windows can achieve higher accuracy than that of fixed windows.

3.3.3 Effectiveness of Unsupervised Methods

Although supervised methods achieve high accuracy, especially on the HDFS data, these methods are not necessarily applicable in a practical setting, where data labels are often not available. Unsupervised anomaly detection methods are proposed to address this problem. To explore the anomaly detection accuracy of unsupervised methods, we evaluate them on the HDFS data and BGL data. As indicated in the last section, the sliding window can lead to more accurate anomaly detection. Therefore, we only report the results of sliding windows on BGL data. As log clustering is extremely time-consuming on HDFS data with around half-a-million instances, tuning parameters become impractical. We then choose the largest log size that we can handle in a reasonable time to represent our HDFS data.
In Figure 3.8, we can observe that all unsupervised methods show good accuracy on HDFS data, but they obtain relatively low accuracy on BGL data. Among three approaches, invariants mining achieves superior performance (with F-measure of 0.91) against other unsupervised anomaly detection methods on both data. Invariants mining automatically constructs linear



(a) Accuracy on HDFS data (b) Accuracy on BGL data

Figure 3.8: Accuracy of unsupervised methods on both datasets.

correlation patterns to detection anomalies, which fit well with the nature BGL data, where failures are marked through some critical events. Log clustering and PCA do not obtain good detection accuracy on BGL data. The poor performance of log clustering is caused by the high-dimensional and sparse characteristics of feature space. As such, log clustering is difficult to separate anomalies and normal instances, often leading to a lot of false positives.

Finding 3: Unsupervised methods generally achieve inferior performance against supervised methods. But invariants mining manifests as a promising method with stable, high performance.

We study in-depth to understand why the PCA does not achieve high accuracy on BGL data. The criterion for PCA to detect anomalies is the distance to normal space (squared prediction error). As Figure 3.9 illustrates, when the distance exceeds a specific threshold (the red dash line represents our current threshold), an instance is identified as an anomaly. However, by

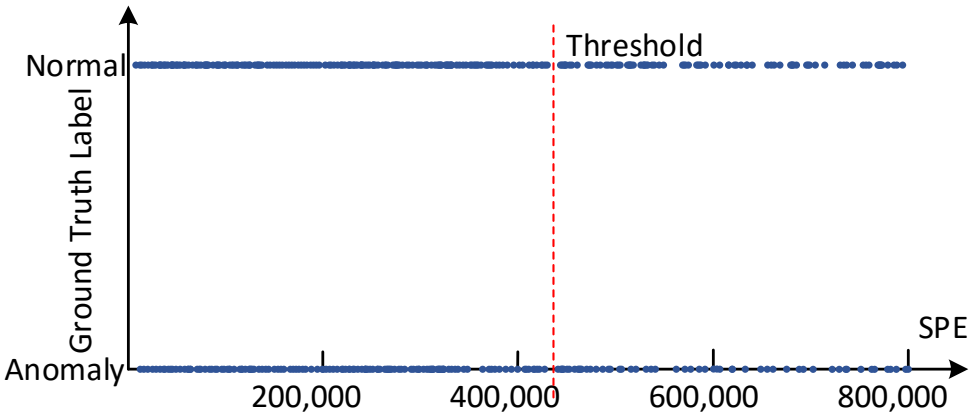
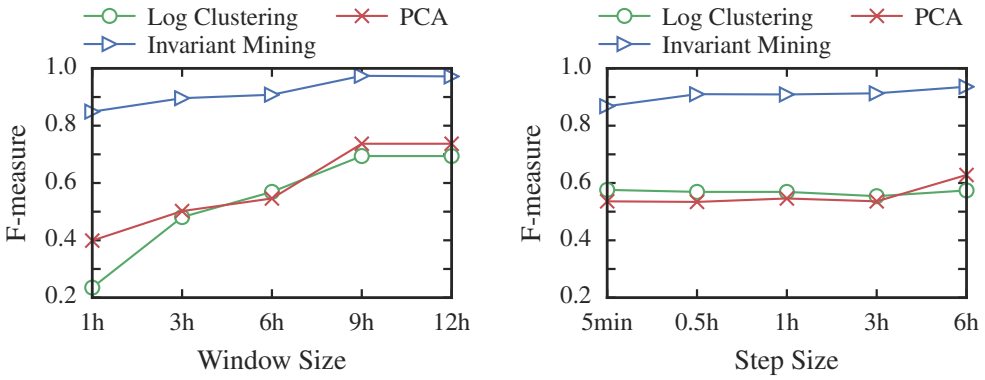


Figure 3.9: Distance distribution in the anomaly space of PCA.

using the ground truth labels to plot the distance distribution, as shown in Figure 3.9, we found that any single threshold cannot naturally separate both classes (normal and abnormal). Therefore, PCA does not perform well on the BGL data.

Like supervised methods, we also conduct experiments on different window size and step size settings to explore their effects on accuracy. As shown in Figure 3.10, we have an interesting observation that the accuracy steadily rises when the window size increases, while the change of step size has little influence on effectiveness. This observation is contrary to what we found for supervised methods. As illustrated in Table 3.2, the window number sharply decreases when the window size increases. Under the large window size setting, more noises are included when more information is covered. However, unsupervised methods could discover more accurate patterns for anomaly detection.

Finding 4: The settings of window size and step size have different effects on supervised methods and unsupervised methods.



(a) Window sizes

(b) Step sizes

Figure 3.10: Accuracy of unsupervised methods with different window sizes and step sizes on BGL data.

3.3.4 Efficiency of Anomaly Detection Methods

In Figure 3.11, the efficiency of all these anomaly detection methods is evaluated on both datasets with varying log sizes. Besides, supervised methods can detect anomalies quickly (less than one minute), while unsupervised methods are much more time-consuming (except PCA). We can observe that all anomaly detection methods scale linearly as the log size increases, except for the log clustering, whose time complexity is $O(n^2)$. Note that both horizontal and vertical axes are not on a linear scale. Furthermore, log clustering cannot handle large-scale datasets in an acceptable time; thus, running time results of log clustering are not fully plotted. It is worth noting that the running time of invariants mining is larger than log clustering on BGL data but not on HDFS data. The reason is that more log events in BGL data than HDFS data increase the time for invariants mining. Besides, we also find that the running time of invariants mining slightly decreases at the log size of 125 megabytes on BGL data.

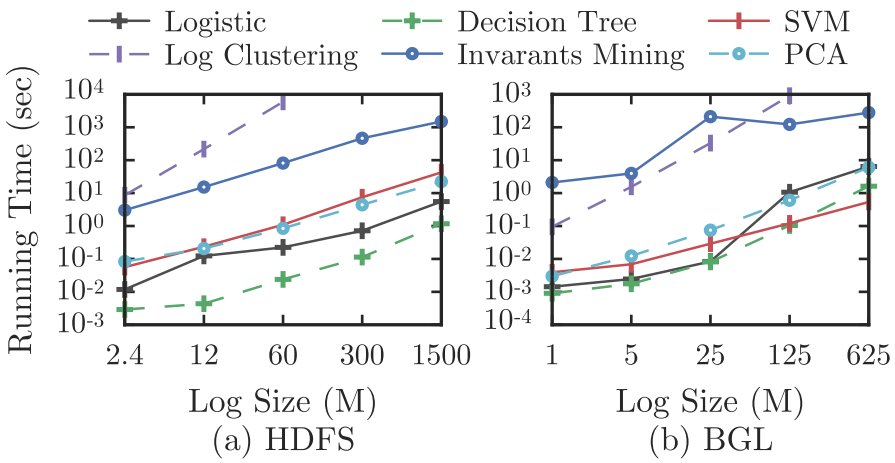


Figure 3.11: Running time for model training with different log data sizes.

We set the stopping criteria to control its brute force searching process on large datasets, which could avoid the unnecessary search for high-dimensional correlations.

Finding 5: Most anomaly detection methods scale linearly with log size, but the methods of Log Clustering and Invariants Mining need further optimizations for speedup.

3.4 Discussion

In this section, we discuss some limitations and related studies of this work and provide some potential directions for future research.

Diversity of datasets Publicly-available log datasets are scarce resources because companies are often unwilling to open their log data due to confidential issues. Thanks to the support from [112, 159], we obtained two production log datasets that

enabled our work. The datasets represent logs from two different systems, but the dataset diversity may still limit the evaluation results and the findings. The availability of more log datasets would allow us to generalize our findings and drastically support related research. It is our future plan to collect more log datasets from open platforms.

Feature representation To generalize our implementations of anomaly detection methods, we focus mainly on a feature space denoted by feature vectors, which has been employed in most of the existing work (e.g., [93, 159]). There are still other features that need further exploration, such as the timestamp of a log message, and the order information of a log sequence can be extracted. However, as reported in [93], logs generated by modern distributed systems are usually interleaved by different processes. Thus, it becomes a great challenge to extract reliable temporal features from such logs.

Other available methods We have reviewed and implemented most of the commonly-used and representative log analysis methods for anomaly detection. However, some other approaches are employing different techniques, such as frequent sequence mining [55], finite state machine [53], and formal concept analysis [40]. We also believe that more are coming out because of the practical importance of log analysis. It is our ongoing work to implement and maintain a more comprehensive set of open-source tools.

Other empirical study Recently, many empirical studies [163, 168, 102] on software reliability emerge because empirical study can usually provide useful and practical insights for both researchers and developers. Yuan et al. [163, 54] study the logging practice of open-source systems and industrial systems, and

they provide improvement suggestions for developers. Pecchia et al. [116] study the logging objectives and issues impacting log analysis in industrial projects. Our study aims at reviewing and benchmarking the existing work that applies log analysis techniques to system anomaly detection.

Open-source log analysis tools There is currently a lack of publicly-available log analysis tools that could be directly utilized for anomaly detection. We also note that a set of new companies (e.g., [3, 4]) are offering log analysis tools as their products, but they are all working as a black box. It would lead to increased difficulty in reproducible research and slow down the overall innovation process. We hope our work makes the first step towards making source code publicly available, and we advocate more efforts in this direction.

Potential directions *1) Model interpretability* Most of current log-based anomaly detection methods are built on machine learning models (such as PCA). But most of these models work as a "black box," and they are hard to interpret to provide intuitive insights, and developers often cannot figure out what the anomalies are. Methods that could reflect the natures of anomalies are highly desired. *2) Real-time log analysis* Current systems and platforms often generate logs in real-time and in huge volumes. Thus, it becomes a big challenge to deal with big log data in real-time. The development of log analysis tools on big data platforms and the functionality of real-time anomaly detection are in demand.

3.5 Summary

In this chapter, we fill the industry and academia gap by providing a detailed review and evaluation of six state-of-the-art anomaly detection methods, including three supervised and three unsupervised methods. We compare their accuracy and efficiency on two representative production log datasets under various settings. Furthermore, we release an open-source toolkit of these anomaly detection methods for easy reuse and further study.

□ **End of chapter.**

Chapter 4

Log-based Interpretation for Problem Identification

Unlike the anomaly detection in Chapter 3, in this chapter, we propose to identify different types of impactful problems using the interpretable logs. For a cloud-based online system that provides 24/7 service, a vast number of logs could be generated every day. However, these logs are highly imbalanced, which hinders the identification of real problems. Besides, those impactful problems should be reported and fixed by engineers with a high priority. To tackle these challenges, we propose Log3C, a novel clustering-based approach to promptly and precisely identify impactful system problems, by utilizing both log sequences (a sequence of log events) and system KPIs. Specifically, we design a novel cascading clustering algorithm and correlate the clusters of log sequences with system KPIs. Experimental results on real-world log data confirm its effectiveness and efficiency.

4.1 Problems and Motivation

Service quality is vital to cloud-based online service systems such as Microsoft Azure, Amazon AWS, Google Cloud, where

logs are frequently utilized in the system maintenance and diagnosis for its interpretable runtime information. Clearly, manual problem diagnosis with logs is very time-consuming and error-prone due to the increasing scale and complexity of large-scale systems. As aforementioned in Chapter 3, a stream of methods based on machine learning have been proposed for log-based problem identification and troubleshooting. Some use supervised methods, such as classification algorithms [162], to categorize system problems. However, they require a large number of labels and substantial manual labeling effort. Others use unsupervised methods, such as PCA [159] and Invariants Mining [93], to detect system anomalies.

However, these approaches can only recognize whether there is a problem or not but cannot distinguish among different types of problems. To identify different problem types, clustering is the most pervasive method [90, 39, 38, 35]. However, it is hard to develop an effective and efficient log-based problem identification approach through clustering. We present the challenges in the following part.

Firstly, large-scale online service systems such as those of Microsoft and Amazon, often run on a 7×24 basis and support hundreds of millions of users, which yields an incredibly large quantity of logs. For instance, a Microsoft service system that we studied can produce dozens of Terabytes of logs per day. Notoriously, conducting conventional clustering on data of such order-of-magnitude consumes a great deal of time, which is unacceptable in practice [77, 5, 48, 65]. Hence, in this chapter, we propose an efficient and effective clustering algorithm to tackle the challenge.

Secondly, there are many types of problems associated with the

logs, and clustering alone cannot determine whether a cluster reflects a problem or not. In previous work on log clustering, developers are required to manually verify the problems during the clustering process [90], which is tedious and time-consuming. Besides, among all the problems, some are impactful and should be fixed with a higher priority. Intuitively, impactful problems can lead to the degradation of system KPIs (Key Performance Indicators), which are widely adopted in industry, such as service availability, average request latency, and failure rate. They measure the health status of a system over a time interval and are collected periodically. A lower KPI value indicates that some system problems may have occurred, and consequently, the service quality deteriorates.

To tackle the second challenge, we leverage both log and KPI data to guide the identification of impactful problems. In practice, systems continuously generate logs, but the KPI values are periodically collected. We use *time interval* to denote the KPI collection frequency. The time interval is typically 1 hour or more, set by the production team. In our setting, we use *failure rate* as the KPI, which is the ratio of failed requests to all requests within a time interval. In each time interval, there could be many logs but only one KPI value (e.g., one failure rate).

Thirdly, log data is highly imbalanced. In a production environment, a well-deployed online service system operates normally most of the time. That is, most of the logs record normal operations, and only a small percentage of logs are problematic and indicate impactful problems. The imbalanced data distribution can severely impede the accuracy of the conventional clustering algorithm [160]. Furthermore, it is intrinsic that some

problems may arise less frequently than others; therefore, these rare problems emerge with fewer log messages. As a result, it is challenging to identify all problem types from the highly imbalanced log data.

For a well-deployed online service system, it operates normally in most cases and exhibits problems occasionally. However, it does not imply that problems are easy to identify. On the contrary, problems are hidden among a vast number of logs, while most logs record the system's normal operations. In addition, various types of service problems may manifest different patterns, occur at different frequencies, and affect the service system in different manners.

As an example, Figure 4.1 shows the long tail distribution of 18 types of log sequences (in logarithmic scale for easy plotting), which are labeled by engineers from product teams. The first two types of log sequences occupy more than 99.8% of the total occurrences ("head") and are generated by normal system operations. The remaining ones indicate different problems, but they all together only take up less than 0.2% of all occurrences ("long tail"). Besides, the occurrences of distinct problem types vary significantly. For example, the first type of problem (the 3rd bar in Figure 4.1) is a "SQL connection problem," which shows that the server cannot connect a SQL database. The most frequent problem occurs over 100 times more often than the least frequent one. The distribution is highly imbalanced and exhibits strong long-tail property, which poses challenges for log-based problem identification.

In this chapter, we propose a novel problem identification framework, Log3C, using log data and system KPI data. To be specific, we propose a novel clustering algorithm, Cascading

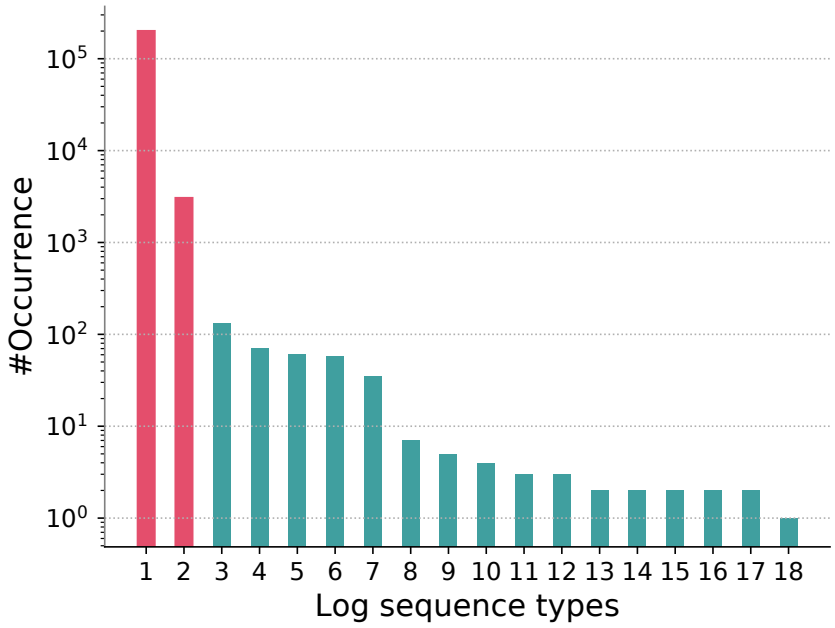


Figure 4.1: An example of long tail distribution in log sequences.

Clustering, which clusters a massive amount of log data by iteratively sampling, clustering, and matching log sequences (sequences of log events). Cascading clustering can significantly reduce the clustering time while keeping high accuracy. Further, we analyze the correlation between log clusters and system KPIs. By integrating the ***C****ascading* ***C****lustering* and ***C****orrelation analysis*, Log3C can promptly and precisely identify impactful service problems. The implementation is available on Github[1]. We evaluate our approach on real-world log data collected from a deployed online service system at Microsoft. The results show that our method can accurately find impactful service problems from large log datasets with high time performance. Log3C can precisely find out problems with an average precision of 0.877 and an average recall of 0.883. We have also successfully applied Log3C to the maintenance of many actual online service systems

[1] https://github.com/logpai/Log3C

at Microsoft.

4.2 Methodology

In this section, we aim at solving the following problems:
1) Given system logs and KPIs, how to detect impactful service system problems automatically? 2) How to identify different kinds of impactful service system problems precisely and promptly?
To this end, we propose the Log3C framework, as depicted in Figure 4.2. Log3C consists of four steps: log parsing, sequence vectorization, cascading clustering, and correlation analysis. In short, at each time interval, logs are parsed into log events, vectorized into sequence vectors, and grouped into multiple clusters through cascading clustering. However, we still cannot extrapolate whether a cluster is an impactful problem, which necessitates the use of KPIs. Consequently, in step four, we correlate clusters and KPIs over different time intervals to find impactful problems. More details are presented in the following sections.

4.2.1 Log Parsing

As aforementioned, log parsing extracts the log event for each raw log message. In this chapter, we use an automatic log parsing method proposed in [53] to extract log events. Following this method, firstly, some common parameter fields (e.g., IP address), are removed using regular expressions. Then, we group log messages into coarse-grained groups based on weighted edit distance. These groups are further split into fine-grained groups

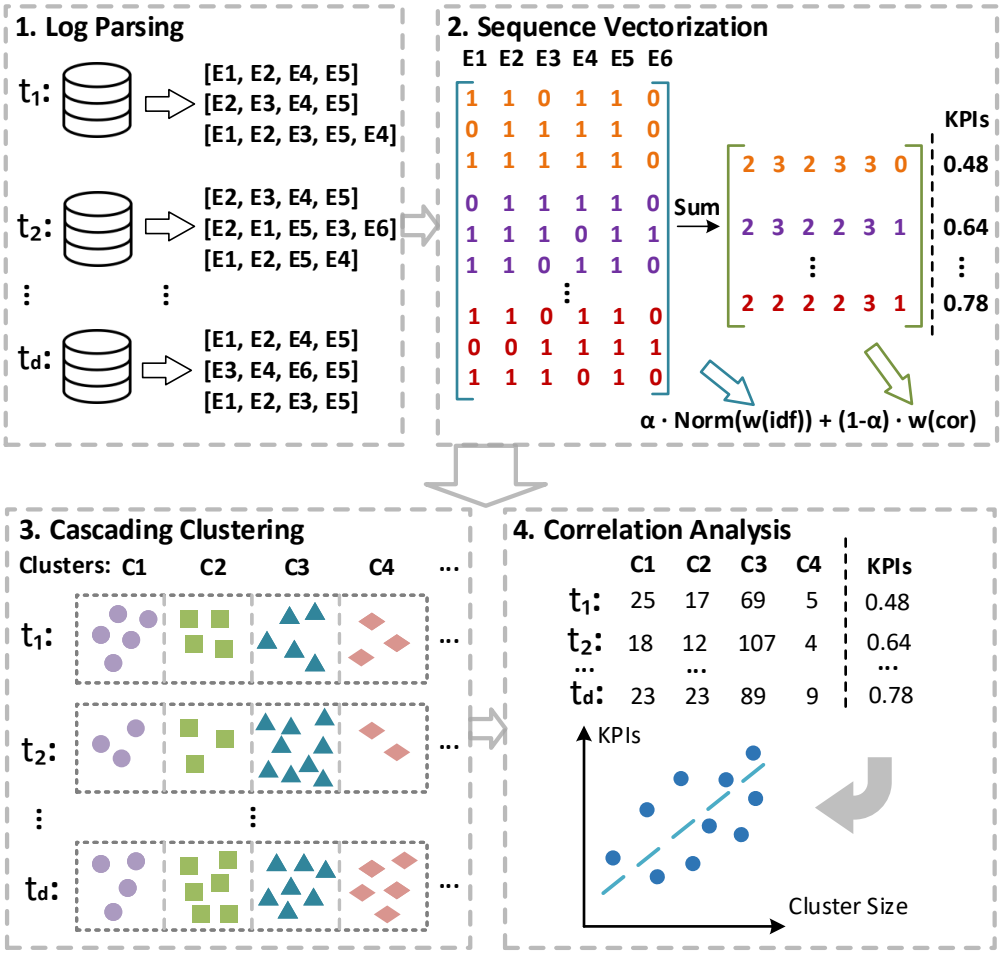


Figure 4.2: An overall framework of Log3C.

of log messages. Finally, a log event is obtained by finding the longest common substrings for each raw log message group.
To form a log sequence, log messages that share the same task ID are linked together and parsed into log events. Moreover, we remove the duplicate events in the log sequence. Generally, repetition often indicates retrying operations or loops, such as continuously trying to connect to a remote server. Without removing duplicates, similar log sequences with different occurrences of the same event are identified as distinct sequences, although they essentially indicate the same system behavior/operation. Following the common practice [90, 130] in log analysis, we remove the duplicate log events.

4.2.2 Sequence Vectorization

After obtaining log sequences from logs in all time intervals, we compute the vector representation for each log sequence. We believe that different log events have different discriminative power in problem identification. As delineated in Step 2 of Figure 4.2, to measure the importance of each log event, we calculate the event weight by combining the following two techniques:
IDF Weighting: IDF (Inverse Document Frequency) is widely utilized in text mining to measure the importance of words in some documents, which lowers the weight of frequent words while increasing rare words' weight [125, 99]. In our scenario, events that frequently appear in numerous log sequences cannot distinguish problems well because problems are relatively rare. Hence, the event and log sequence are analogous to word and document, respectively. We aggregate log sequences in all time intervals to calculate the IDF weight, which is defined in the

first equation, where N is the total number of all log sequences, and n_e is the number of log sequences that contain the event e. With IDF weighting, frequent events have low weights, while rare log events are weighted high.

$$w_{idf}(e) = \log\left(\frac{N}{n_e}\right)$$
$$w(e) = \alpha * Norm(w_{idf}(e)) + (1-\alpha) * w_{cor}(e)$$

Importance Weighting: In problem identification, it is intuitive that events strongly correlate with KPI degradation are more critical and should be weighted more. Therefore, we build a regression model between log events and KPI values to find the importance weight. To achieve so, as shown in Figure 4.2, in each time interval, we sum the occurrence of each event in all log sequences (three in the example) as a summary sequence vector. After that, we get d summary sequence vectors, and d KPI values are also available as aforementioned. Then, a multivariate linear regression model is applied to evaluate the correlation between log events and KPIs. The weights $w_{cor}(e)$ obtained from the regression model serve as the importance weights for log events e. Note that the regression model only aims to find the importance weight for the log event.

As denoted in the second equation, the final event weight is the weighted sum of IDF weight and importance weight. Besides, we use *Sigmoid* function [158] to normalize the IDF weight into the range of $[0, 1]$. Since the importance weight is directly associated with KPIs and is thereby more effective in problem identification, we value the importance weight more, i.e., $\alpha < 0.5$. In our experiments, we empirically set α to 0.2. Given the final event weights, the weighted sequence vectors can

be easily obtained. For simplicity, hereafter, we use "sequence vectors" to refer to "weighted sequence vectors". Note that each log sequence has a corresponding sequence vector.

4.2.3 Cascading Clustering

Once all log sequences are vectorized, we group sequence vectors into clusters separately for each time interval. However, the conventional clustering methods are incredibly time-consuming when the data size is large [77, 5, 48, 65] because distances between any pair of samples are required. As mentioned in Section 4.1, log sequences follow the long tail distribution and are highly imbalanced. Based on the observation, we propose a novel clustering algorithm, *cascading clustering*, to group sequence vectors into clusters (different log sequence types) promptly and precisely, where each cluster represents one kind of log sequence (system behavior).

Figure 4.3 depicts the procedure of cascading clustering, which leverages iterative processing, including sampling, clustering, matching. The input of cascading clustering is all the sequence vectors in a time interval, and the output is a number of clusters. To be more specific, we first sample a portion of sequence vectors, on which a conventional clustering method (e.g., hierarchical clustering) is applied to generate multiple groups. Then, a pattern can be extracted from each group. In the matching step, we match all the *original* sequence vectors with the patterns to determine their cluster. Those mismatched sequence vectors are collected and fed into the next iteration. By iterating these processes, all sequence vectors can be clustered rapidly and accurately. The reason is that large clusters are separated from the remaining data at the first several iterations.

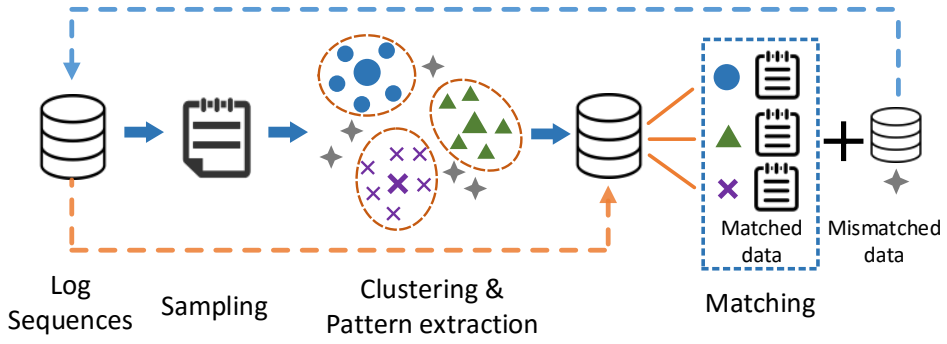


Figure 4.3: An overview of the cascading clustering algorithm.

1) Sampling

Given numerous sequence vectors in each time interval, we first sample a portion of them through Simple Random Sampling (SRS). Each sequence vector has an equal probability of p (e.g., 0.1%) to be selected. Suppose there are N sequence vectors in the input data, then the sampled data size is $M = \lceil p * N \rceil$. After sampling, log sequence types (clusters) that dominate in the original input data are still dominant in the sampled data.

2) Clustering

After sampling M sequence vectors from the input data, we group these sequence vectors into multiple clusters and extract a representative vector (pattern) from every cluster. To do so, we calculate the distance between every two sequence vectors and apply an ordinary clustering algorithm.

Distance Metric: During clustering, we use Euclidean distance as the distance metric, which is defined in the following equations: u and v are two sequence vectors, and l is the vector length, which is the number of log events. u_i and v_i are the i-th value in vector u and v, respectively.

$$d(u,v) = \sqrt{\|u-v\|} = \sqrt{\sum_{i=1}^{l}(u_i - v_i)^2}$$

Clustering Technique: We utilize *Hierarchical Agglomerative Clustering (HAC)* to conduct clustering. At first, each sequence vector itself forms a cluster, and the closest two clusters are merged into a new one. To find the closest clusters, we use the complete linkage [156] to measure the cluster distance, as shown in the following:

$$D(A,B) = max\{d(a,b), \quad \forall a \in A, \forall b \in B\}$$

D is the cluster distance between two clusters A and B, which is defined as the longest distance between any two elements (one in each cluster) in the clusters. The merging process continues until reaching a distance threshold of θ. That is, the clustering stops when all the distances between clusters are larger than θ. In Section 4.3.4, we also study the effect of different thresholds. After clustering, similar sequence vectors are grouped into the same cluster, while dissimilar sequence vectors are separated into different clusters.

Pattern Extraction: After clustering, a representative vector is extracted for each cluster, which serves as the pattern of a group of similar log sequences. To achieve so, we compute the mean vector [9] of all sequence vectors in a cluster. Assume that there are k clusters, then k mean vectors (patterns) can be extracted to represent those clusters, respectively.

3) Matching

As illustrated in Figure 4.3, we match each sequence vector in the *original* input data (of size N) to one of the k patterns,

which are obtained by clustering M sampled sequence vectors. To this end, for each sequence vector x, we calculate the distance between it and every pattern. Furthermore, we compute the minimum distance of μ as denoted in the following function, where P is a set of all patterns. If the minimum distance μ is smaller than the threshold θ defined in the clustering step, the sequence vector x is matched with a pattern successfully and thereby can be assigned to the corresponding cluster. Otherwise, this sequence vector is classified as mismatched. Note that those unmatched sequence vectors would proceed to the next iteration.

$$\mu = \min\{d(x, P_j), \quad \forall j \in \{1, 2, ..., k\}\}$$

4) Cascading

After going through the above three processes (i.e., sampling, clustering, and matching), most of the data in a time interval can be grouped into clusters. At the same time, some sequence vectors may remain unmatched. Hence, we further process the mismatched data by repeating the above-mentioned procedures. During each iteration, new clusters are grouped based on current mismatched data, new patterns are extracted, and new mismatched data are produced. In our experiments, we cascade these repetitions until all the sequence vectors are successfully grouped together.

Algorithm and Time Complexity Analysis. The pseudo-code of *Cascading clustering* is described in the Algorithm 1. The algorithm takes sequence vectors in a time interval as input data, with sample rate and clustering distance threshold as hyper-parameters. After cascading clustering, the algorithm

Algorithm 1: Cascading Clustering

Input : Sequence vector data D, Sample rate p, Clustering threshold θ

Output: Sequence clusters $globalClusters$, Pattern set $gloablPatList$

```
misMatchData = D;
globalPatList = ∅; globalClusters = ∅;
while misMatchData! = ∅ do
    SampleData = ∅;
    /* Sampling sequence vectors                          */
    foreach seqVec ∈ D do
        if random(0, 1) <= p then
            SampleData.append(seqVec);
        end
    end
    /* Hierarchical clustering                            */
    localClusters = HAC(SampledData, θ);
    localPatList = patternExtraction(clusters);
    /* Matching and finding mismatched data               */
    newMismatchData = ∅;
    foreach seqVec ∈ misMatchData do
        foreach pat ∈ localPatList do
            distList.append(dist(seqVec, pat));
        end
        if min(distList) > θ then
            newMismatchData.append(seqVec);
        end
    end
    misMatchData = newMismatchData;
    globalPatList.extend(localPatList);
    globalClusters.extend(localClusters);
end
Return globalClusters, globalPatList
```

outputs all sequence vector clusters and a set of patterns. To initialize, we assign all sequence vectors D to the mismatched data. Besides, we define two global variables (lines 1-2) to store the clusters and patterns. Then, the sampled data is obtained with a sampling rate of p (lines 3-10). In lines 12 and 13, we perform the hierarchical agglomerative clustering (HAC) on sampled data with a threshold θ and extract the cluster patterns. Other clustering methods (e.g., K-Means) are also applicable here. During the matching process (line 16-23), we use *distList* (line 17-19) to store the distances between a sequence vector and every cluster pattern. The sequence vector is allocated to the closest cluster if the distance is smaller than the threshold θ. The remaining mismatched data is updated (lines 24) and processed in the next cascading round.

We now analyze the time complexity of the proposed algorithm. Note that only the core parts of the cascading clustering algorithm are considered, i.e., distance calculation and matching, because they consume most of the time. We set the data size to N, which is a large number (e.g., larger than 10^6). The sample rate p is usually a user-defined small number (e.g., less than 1%). For standard hierarchical agglomerative clustering, the distance calculation takes $O(N^2)$ time complexity, and no matching is involved. For cascading clustering, suppose that pN data instances are selected and clustered into k_1 groups firstly, and further N_1 instances are mismatched. Therefore, the time complexity of the first round is $T_1 = p^2N^2 + k_1N$. After t iterations, the total number of clusters is $K = \sum_{i=1}^{t} k_i$. Therefore, the overall time complexity $T(cc)$ is calculated as:

$$
\begin{aligned}
T(cc) &= p^2N^2 + k_1N + p^2N_1^2 + k_2N_1 + ... + p^2N_t^2 + k_tN_{t-1} \\
&= p^2N^2 + \sum_{i=1}^{t} p^2N_i^2 + k_1N + \sum_{i=1}^{t-1} k_{i+1}N_i \\
&< p^2N^2 + tp^2N_1^2 + KN < (pN + p\sqrt{t}N_1 + \sqrt{KN})^2
\end{aligned}
$$

Since N is a large number and K is the total number of clusters, we have $K \ll N$ and $\sqrt{KN} \ll N$. Because the data follows the long tail distribution, and the "head" occupies most of the data (e.g., more than 80%). After several iterations, most data can be successfully clustered and matched. Recall that $p \ll 1$, we then have $p\sqrt{t}N_1 \ll N$ and $pN \ll N$. Therefore, the inequality $pN + p\sqrt{t}N_1 + \sqrt{KN} < N$ holds and the left-hand side is much smaller. Given that $f(X) = X^2$ is a monotonic increasing function ($X \geq 0$), where $f(X)$ decreases with the decreasing of X. We then have $(pN + p\sqrt{t}N_1 + \sqrt{KN})^2 \ll N^2$ satisfied, which indicates that cascading clustering consumes much less time than standard clustering in terms of distance calculation and matching. In our experiments, we empirically evaluate the time performance of cascading clustering, and the results support our theoretical analysis.

4.2.4 Correlation Analysis

As described in Figure 4.2, log sequence vectors are grouped into multiple clusters separately in each time interval. These clusters only represent different types of log sequences (system behaviors) but may not necessarily be problems. From the groups, we identify impactful problems that could lead to the degradation of KPI. Intuitively, KPI degrades more if impact

problems occur more frequently. Hence, we aim to identify those clusters that highly correlate with KPI's changes. To do so, we model the correlation between cluster sizes and KPI values over multiple time intervals. Unlike the importance weighting in Section 4.2.2 that discriminates the importance of different log events, this step attempts to identify impactful problems from clusters of sequence vectors.

More specifically, we utilize a multivariate linear regression (MLR) model (as shown in the following equation), which correlates independent variables (cluster sizes) with the dependent variable (KPI). Among all independent variables, those have statistical significance make notable contributions to the dependent variable. Moreover, the corresponding clusters indicate impactful problems, whose occurrences contribute to the change of KPI. Statistical significance is widely utilized in the identification of important variables [132, 74].

$$\begin{bmatrix} c_{11} & c_{12} & c_{13} & \dots & c_{1n} \\ c_{21} & c_{22} & c_{23} & \dots & c_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_{d1} & c_{d2} & c_{d3} & \dots & c_{dn} \end{bmatrix} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_d \end{bmatrix} = \begin{bmatrix} KPI_1 \\ KPI_2 \\ \vdots \\ KPI_d \end{bmatrix}$$

In the above equation, suppose there are n clusters generated during d time intervals. c_{ij} represents the cluster size (the number of sequence vectors) of the j-th cluster at the time interval i. KPI_i is the system KPI value at the time interval i. β_i and ε_i are coefficients and error terms that would be learned from data.

To find out which clusters highly correlate with the KPI values, we adopt the *t-statistic*, which is a widely used statistical hypothesis test. In our MLR model, important groups (indi-

cating impactful problems) make significant contributions to the change of KPIs, and their coefficients should not be zero. Therefore, we make a null hypothesis for each independent variable that its coefficient is zero. A two-tailed t-test is then applied to measure each coefficient's significant difference, i.e., the probability p that the null hypothesis is true. A lower *p-value* is preferred since it represents a higher likelihood of rejecting the null hypothesis. If *p-value* is less than or equal to a given threshold of α (significance level), the corresponding cluster implies an impactful problem that affects the KPI. In this chapter, we set α to 0.05, a common setting in the hypothesis test.

4.3 Evaluations

In this section, we evaluate our approach using real-world data from industry. We aim at answering the following research questions:
RQ1: How effective is the proposed Log3C approach in detecting impactful problems?
RQ2: How effective is the proposed Log3C approach in identifying different types of problems?
RQ3: How does cascading clustering perform under different configurations?

4.3.1 Experimental Setup

Datasets: We collect real-world data from an online service system X of Microsoft. Service X is a large scale online service system that serves hundreds of millions of users globally on a 24/7 basis. Service X has been running over the years and

has achieved high service availability. The system operates in multiple data centers with a large number of machines, each of which produces a vast quantity of logs every hour. Service X utilizes the load balancing strategies, and end-user requests are accepted and dispatched to different back-ends. There are many components at the application level, and each component has its specific functionalities. Most user requests involve multiple components on various servers. Each component generates logs, and all the logs are uploaded to a distributed HDFS-like data storage automatically. Each machine or component has a probability of failing, leading to various problem types. We use *failure rate* as the KPI, which shows the percentage of failed requests in a time interval.

Data	Snapshot starts	#Log Seq (Size)	#Events	#Types
Data 1	Sept 5th 10:50	359,843 (722MB)	365	16
Data 2	Oct 5th 04:30	472,399 (996MB)	526	21
Data 3	Nov 5th 18:50	184,751 (407MB)	409	14

Table 4.1: Summary of service X log data.

	Method	Precision	Recall	F1-Measure
Data 1	PCA	0.465	0.946	0.623
	IM	0.604	1.000	0.753
	Log3C	**0.900**	0.920	**0.910**
	Method	Precision	Recall	F1-Measure
Data 2	PCA	0.142	0.834	0.242
	IM	0.160	0.847	0.269
	Log3C	**0.897**	0.826	**0.860**
	Method	Precision	Recall	F1-Measure
Data 1	PCA	0.207	0.922	0.338
	IM	0.168	0.704	0.271
	Log3C	**0.834**	0.903	**0.868**

Table 4.2: Accuracy of problem detection on service X data.

Service X produces a large quantity of log data consisting of billions of log messages. However, it is unrealistic to evaluate Log3C on all the data due to the lack of labels. The labeling difficulties origin from two aspects: first, the log sequences are enormous. Second, various problem types can exist, and human labeling is very time-consuming and error-prone. Therefore, we extract logs that were generated during a specified period[2] on three different days. In this way, three real-world datasets (i.e., Data 1, 2, 3) are obtained, as shown in Table 4.1. Besides the log data, we also collect the corresponding KPI values. During labeling, product team engineers utilize their domain knowledge to identify the normal log sequences. Then, they manually inspect the rest log sequences from two aspects: 1) Does the log sequence indicate a problem? 2) What is the problem type? Table 4.1 shows the number of problem types identified in the evaluation datasets. Note that the manual labels are only used for evaluating the effectiveness of Log3C in our experiments. Log3C is an unsupervised method, which only requires log and KPI data to identify problems.

Implementation and Environments: We use Python to implement our approach for easy comparison, and run the experiments on a Windows Server 2012 (Intel(R) Xeon(R) CPU E5-4657L v2 @ 2.40GHz 2.40 with 1.00TB Memory).

Evaluation Metrics: To measure the effectiveness of Log3C in problem detection, we use precision, recall, and F1-measure. Given the labels from engineers, we calculate these metrics as follows:

Precision: the percentage of log sequences that are correctly identified as problems over all the log sequences that are

[2]The actual period is anonymous due to data sensitivity.

identified as problems: Precision $= \frac{TP}{TP+FP}$.
Recall: the percentage of log sequences that are correctly identified as problems over all problematic log sequences: Recall $= \frac{TP}{TP+FN}$.
F1-measure: the harmonic mean of *precision* and *recall*.
TP is the number of problematic log sequences that are correctly detected by Log3C, *FP* is the number of non-problematic log sequences that are wrongly identified as problems. *FN* is the number of problematic log sequences that are not detected by Log3C, *TN* is the number of log sequences that are identified as non-problematic by both engineers and Log3C.
To measure the effectiveness of clustering, we use the *Normalized Mutual Information* (NMI), which is a widely used metric for evaluating clustering quality [137]. The value of NMI ranges from 0 to 1. The closer to 1, the better the clustering results. To measure the efficiency of cascading clustering, we record the total time (in seconds) spent on clustering.

4.3.2 Effectiveness in Problem Detection

To answer RQ1, we apply Log3C to the three datasets collected from Service X and evaluate the precision, recall, and F1-measure. The results are shown in Table 4.2. Log3C achieves satisfactory accuracy, with recall ranging from 0.826 to 0.92 and precision ranging from 0.834 to 0.9. The F1-measures on the three datasets are 0.91, 0.86, and 0.868, respectively.
Furthermore, we compare our method with two typical methods: PCA [159] and Invariants Mining [93]. All these three methods are unsupervised, log-based problem identification methods. PCA projects the log sequence vectors into a subspace. If the projected vector is far from the majority, it is considered

as a problem. Invariants Mining extracts the linear relations (invariants) between log event occurrences, which hypothesizes that log events are often pairwise generated. For example, when processing files, "File A is opened" and "File A is closed" should be printed as a pair. Log sequences that violate the invariants are regarded as problematic.

Log3C achieves good recalls (similar to those obtained by two comparative methods) and surpasses the comparative approaches concerning precision and F1-measure. The absolute improvement in F1-measure ranges from 15.7% to 61.8% on the three datasets. The two comparative methods all achieve low precision (less than 0.61), while the precisions achieved by Log3C are higher than 0.83. We also explore the reasons for the low precision of the competing methods. In principle, PCA and invariants mining aim at finding the abnormal log sequences from the entire data. However, some rare user/system behaviors can be wrongly identified as problems. Thus, many false positives are generated, which result in high recall and low precision. More details are described in Section 4.4.4.

Regarding the time usage of problem detection, on average, it takes Log3C 223.93 seconds to produce the results for each dataset, while PCA takes around 911.97 seconds, and invariants mining consumes 1830.78 seconds. The time performance of Log3C is satisfactory, considering a large amount of log sequence data.

4.3.3 Effectiveness in Problem Identification

In Log3C, we propose a novel cascading clustering algorithm to group log sequences into clusters that represent different types of problems. For the ease of evaluation, clusters that represent nor-

mal system behaviors are considered as special "non-problem" types. In this section, we use NMI to evaluate the effectiveness of Log3C in identifying different kinds of problems. We also compare the performance of cascading clustering (denoted as CC) with the standard clustering method hierarchical agglomerative clustering (denoted as SC). To compare fairly, we implement a variant of Log3C that replaces CC with SC (*Log3C-SC*). All the other settings (e.g., distance threshold, event weight) remain the same.

	Size	10k	50k	100k	200k
Data 1	Log3C-SC	0.659	0.706	0.781	0.822
	Log3C	**0.720**	**0.740**	**0.798**	**0.834**
	Size	10k	50k	100k	200k
Data 2	Log3C-SC	0.610	0.549	0.600	0.650
	Log3C	**0.624**	0.514	**0.663**	**0.715**
	Size	10k	50k	100k	180k
Data 3	Log3C-SC	0.601	0.404	0.792	0.828
	Log3C	**0.680**	**0.453**	**0.837**	**0.910**

Table 4.3: Clustering accuracy on service X data.

Table 4.3 presents the NMI results, in which data size refers to the number of log sequences. We sample four subsets of each dataset with size ranging from 10k to 200k (for Data 3, 180k is used instead of 200k as its total size is around 180k). From the table, we can conclude that Log3C (with cascading clustering) is effective in grouping numerous log sequences into different clusters and outperforms Log3C-SC on all three datasets. Besides, with the increase of data size, clustering accuracy increases. It is because more accurate event weights can be obtained with more data during sequence vectorization. For instance, when 200k data is used, the NMI values achieved by Log3C range from 0.715 to 0.91 (180k for Data 3).

Data 1	Size	10k	50k	100k	200k
	SC	127.6	2319.2	9662.3	38415.5
	CC	1.0	4.3	9.2	20.7
Data 2	Size	10k	50k	100k	200k
	SC	80.6	2469.1	8641.2	38614.0
	CC	0.7	3.8	9.5	18.9
Data 3	Size	10k	50k	100k	180k
	SC	81.5	2417.2	8761.2	33728.3
	CC	0.8	4.0	8.8	18.3

Table 4.4: Efficiency (in seconds) of clustering algorithms.

θ	0.10	0.15	0.20	0.25	0.30	0.35	0.40	0.45
NMI	0.848	0.867	0.912	0.916	0.928	0.898	0.898	0.887

Table 4.5: Cascading clustering accuracy under distance threshold θ.

We also evaluate the time performance of cascading clustering. Table 4.4 shows that our cascading clustering (CC) dramatically saves time in contrast to the standard HAC clustering (SC), and the comparison is more noticeable when the data size grows. For example, our approach is around 1800x faster than standard clustering on dataset 1 with a volume of 200k.

4.3.4 Performance under Different Configurations

In Section 4.2.3, we introduced two crucial hyper-parameters in cascading clustering: the sample rate p and the distance threshold θ. In this section, we evaluate clustering accuracy and time performance under different configurations of parameters. We conduct the experiments on Data 1, but the results are also applicable to other datasets.

Distance threshold θ: We first fix the sample rate (1%) and vary the distance threshold θ for cascading clustering. Table 4.5 illustrates the clustering accuracy (NMI). When θ is 0.30, the

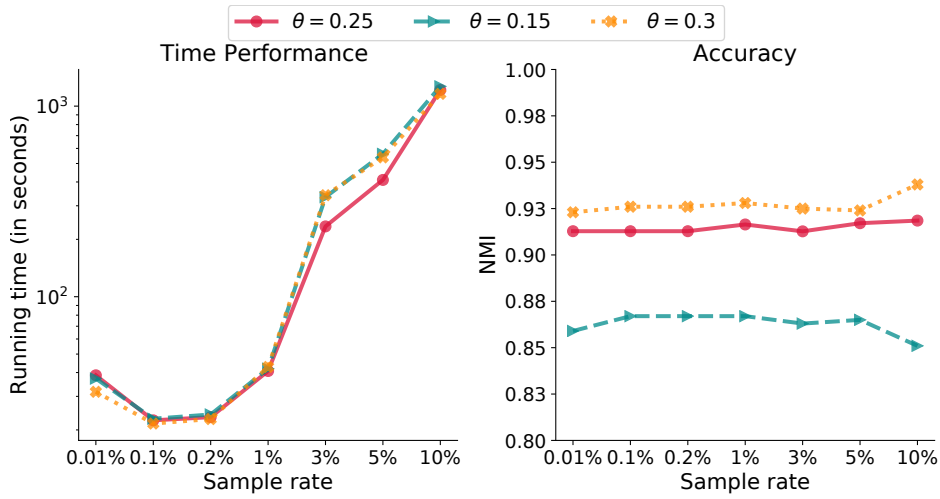


Figure 4.4: Effectiveness and efficiency of cascading clustering under different configurations.

highest NMI value (0.928) is achieved. However, we also observe that NMI changes slightly when the threshold changes within a reasonable range. The results show that our proposed cascading clustering algorithm is insensitive to the distance threshold to some degree.

Sample rate p: It is straightforward that the sample rate can affect the time performance of cascading clustering because it takes more time to do clustering on a larger dataset. To verify it, we change the sample rate while fixing the distance threshold. Figure 4.4 depicts the results. We conduct the experiments with different sample rates under three distance thresholds (0.15, 0.25, and 0.3). The left sub-figure shows that a higher sample rate generally causes more time usage. However, when the sample rate is low, e.g., 0.01%, a little more time is required. A small sample rate leads to more iterations of clustering in cascading clustering, which hampers cascading clustering efficiency. Besides, we also evaluate the clustering accuracy under different sample rates. As shown in the right

sub-figure of Figure 4.4, clustering accuracy (NMI) is relatively stable when the sample rate changes. It indicates that the NMI value floats slightly with a small standard deviation of 0.0071. In summary, we can conclude that a low sample rate generally does not affect clustering accuracy and costs much less time. This finding can guide the setting of the sample rate in practice.

Data Size	20k		50k		100k	
#Clusters	SC	CC	SC	CC	SC	CC
20	0.958	0.947	0.885	0.918	0.837	0.850
50	0.971	0.961	0.937	0.960	0.906	0.916
100	0.976	0.995	0.956	0.979	0.927	0.939
200	0.988	0.990	0.973	0.973	0.955	0.953

Data Size	200k		300k		600k	
#Clusters	SC	CC	SC	CC	SC	CC
20	0.786	0.791	0.758	0.774	-	0.725
50	0.864	0.883	0.837	0.854	-	0.811
100	0.903	0.911	0.889	0.892	-	0.859
200	0.929	0.937	0.914	0.925	-	0.896

Table 4.6: Accuracy of standard clustering (SC) and cascading clustering (CC) on synthetic data.

4.4 Discussion

In this section, we discuss some settings and threats that might affect the performance of our methods. In addition, we share some of our success stories and lessons learned when deploying the Log3C framework.

4.4.1 Discussions of Results

1) Cluster Numbers in Cascading Clustering

In Section 4.3, we explored the efficiency and effectiveness of the cascading clustering algorithm on real-world datasets. In this section, we evaluate our cascading clustering algorithm on some synthetic datasets. More specifically, we generate some synthetic datasets with a different number of clusters.

To simulate a scenario that is similar to problem identification, we synthesize several synthetic datasets consisting of multiple clusters, where the cluster sizes follow the long-tail distribution. In more detail, 1) we firstly synthesize a dataset of multiple clusters, and the data sample dimension is fixed at 200. The data samples in each group follow the multivariate normal distribution [157]. 2) Then, we use the pow law function (i.e., $f(x) = \alpha x^{-k}$) to determine the size of each cluster with some Gaussian noises added, as noises always exist in real data. In this way, we can generate multiple datasets with different data sizes (from 20k to 600k) and various clusters (from 20 to 200). Figure 4.5 shows the time performance (in logarithmic scale for easy plotting) of standard clustering and cascading clustering where the number of clusters is 50 and 200. We also vary the synthetic data size from 20k to 600k. Cascading clustering requires much less time than standard clustering (hierarchical agglomerative clustering), with different cluster numbers. For example, standard clustering takes 11512.2 seconds (around 3.19 hours) on 200k data with 50 clusters, while cascading clustering (sample rate is 1%) only takes 10.3 seconds on the same dataset. Our cascading clustering algorithm is more than 1000× faster than standard clustering.

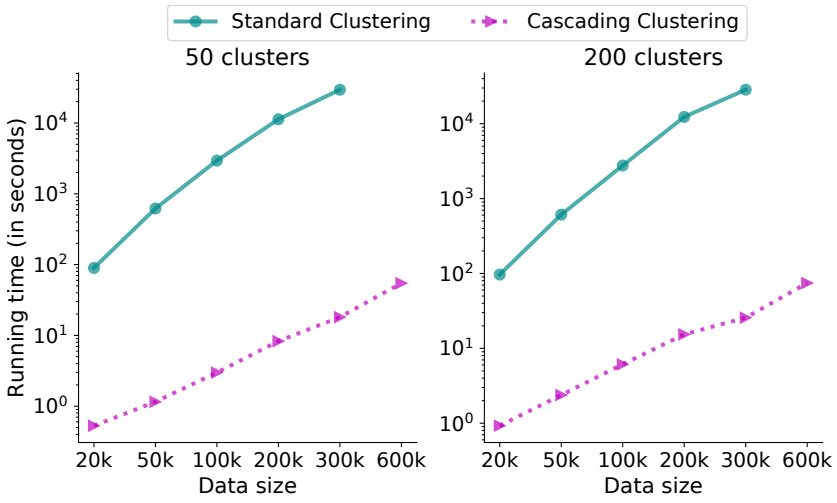


Figure 4.5: Efficiency of clustering methods on synthetic data with 50 cluster (left) and 200 clusters (right).

In Table 4.6, we measure the clustering accuracy (in terms of NMI) under different data sizes and cluster numbers. We can conclude from the table that, overall, cascading clustering leads to equal or slightly better accuracy when compared with the standard clustering. The main reason is that our cascading clustering algorithm is specially designed for long-tailed data. The small clusters can be precisely clustered. Moreover, the evaluation results of standard clustering on 600k data are not available due to the out-of-memory (more than 1TB) computation. From Table 4.6, we can also observe that clustering accuracy increases with the increase of cluster number and decreases when the data size increases.

2) Impact of Log Data Quality

The quality of log data is crucial to log-based problem identification. For a large-scale service system, logs are usually generated on local machines, which are then collected and uploaded to a

data center separately. During the process, some logs may be missed or duplicated. For duplicated logs, they do not affect the accuracy of Log3C as duplicates are removed from log sequences, as described in Section 4.2.2. To evaluate the impact of missing log data, we randomly remove a certain percentage (missing rate) of logs from Data 1 and then evaluate the accuracy of Log3C. We use three different missing rates 0.1%, 0.5%, and 1%. The resulting F1-measures are 0.877, 0.834, 0.600, respectively. We can find that a higher missing rate could lead to a lower problem identification accuracy. Therefore, we suggest ensuring the log data quality before applying Log3C in practice.

4.4.2 Threats to Validity

We have identified the following threats to validities:

Subject Systems: In our experiment, we only collect log data from one online service system (Service X). This system is a typical, large-scale online system, from which sufficient data can be collected. Furthermore, we have applied our approach to the maintenance of the actual online service systems of Microsoft. In the future, we will evaluate Log3C on more subject systems and report the evaluation results.

Selection of KPI: In our experiments, we use *failure rate* as the KPI for problem identification. *failure rate* is an important KPI for evaluating system service availability. There are also other KPIs, such as mean time between failures, average request latency, throughput, etc. In our future work, we will experiment with problem identification concerning different KPI metrics.

Noises in labeling: Our experiments are conducted on three datasets collected as a period of logs on three different days. The engineers manually inspected and labeled the log sequences.

Noises (false positives/negatives) may be introduced during the manual labeling process. However, as the engineers are experienced professionals of the product team who maintain the service system, we believe the amount of noise is small (if it exists).

4.4.3 Success Story

Log3C is successfully applied to Microsoft's Service X system for log analysis. Service X provides hundreds of millions of global end-users online services on a $7 * 24$ basis. For online service systems like Service X, inspecting logs is the only feasible way for fault diagnosis. In the Service X system, more than one Terabyte logs (around billions) is generated in a few hours, and it is a great challenge to process the great volume of logs. A distributed version of Log3C is developed and employed in Service X. Billions of logs can be handled within hours using our method, which helps the service team identify different log sequence types and detect system problems. For example, in April 2015, a severe problem occurred in one component of Service X on some servers. The problem was caused by an incompatibility issue between a patch and a previous product version during a system upgrade. The service team received lots of user complains regarding this problem. Our Log3C successfully detected the problem and reported it to the service team. The service team also utilized Log3C to investigate the logs and precisely identified the problem type. With the help of Log3C, the team quickly resolved this critical issue and redeployed the system.

Log3C is also integrated into Microsoft's Product B, an integrated environment for analyzing the root causes of service

issues. Tens of billions of logs are collected and processed by Product B every day, in which Log3C is the log analysis engine. Using Log3C, Product B divides the log sequences into different clusters and identifies many service problems automatically. Log3C significantly reduces engineers' efforts on manually inspecting the logs and pinpointing root causes of failures. Furthermore, fault patterns are also extracted and maintained for analyzing similar problems in the future.

4.4.4 Lessons Learned

1) Problems != Outliers

Recent research [159, 93] proposed many approaches to detect system anomalies using data mining and machine learning techniques. These approaches work well for relatively small systems. Their ideas are mainly based on the following hypothesis: systems are regular most of the time, and problems are "outliers". Many current approaches try to detect the "outliers" from a massive volume of log data. For example, PCA [159] attempts to map all data to a normal subspace, and those cannot be projected to the normal space are considered as anomalies. However, outliers are not always real problems. Some outliers are caused by certain infrequent user behaviors, e.g., rarely-used system features. Our experiences with the production system reveal that there are indeed many unusual user behaviors, which are not real problems. A lot of effort could be wasted by examining these false positives. In our work, we utilize system KPI to guide the identification of real system problems.

2) The Trend of Problem Reports Is Important

In production, engineers care about the occurrence of a problem and the number of problem reports (i.e., the instances of problems) over time (which reflects the number of users affected by the problem over time). Through our communication with a principal architect of a widely-used service in Microsoft, we conclude three types of important trends: 1) Increasing. When the size of one certain problem continuously increases for a period, the production team should be notified. It is because the number of problem reports may accumulate and cause even serious consequences. 2) The appearance of new problems: when a previously unknown problem appears, it is often a sign of new bugs, which may be introduced by software updates or a newly launched product feature. 3) The disappearance of problems: The disappearing trend is very interesting. In production, after fixing a problem, the scale of the problem is expected to decrease. However, sometimes the disappearing trend may stop at a certain point (the service team continues to receive reports for the same problem), which often indicates an incomplete bug-fix or a partial solution. More debugging and diagnosis work are needed to identify the root cause of the problem and propose a complete bug-fixing solution.

4.5 Summary

Large-scale online service systems generate a vast number of logs, which can be used for troubleshooting. In this chapter, we propose Log3C, a novel framework for automated problem identification via log analysis. At the heart of Log3C is cascading clustering, an innovative clustering algorithm for clustering

a large number of highly imbalanced log sequences. The grouped log sequences are correlated with system KPI through a regression model, from which the clusters that represent impactful problems are identified. We evaluate Log3C using real-world log data. Besides, we also apply our approach to the maintenance of actual online service systems. The results confirm the effectiveness and efficiency of Log3C in practice. In the future, we will apply Log3C to various software systems to further evaluate its effectiveness and efficiency. Also, the proposed Cascading Clustering algorithm is a general algorithm that can be applied to a wide range of problems.

□ **End of chapter.**

Chapter 5

Gradient-based Input-Output Attribution

In this chapter, we focus on the interpretation of intelligent software for its software reliability engineering. To specify, we take the neural machine translation model as intelligent software due to its wide popularity in practice. Although neural machine translation (NMT) has advanced the state-of-the-art on various language pairs, the interpretability of NMT remains unsatisfactory. In this chapter, we propose to address this gap by focusing on understanding the input-output behavior of NMT models. Specifically, we measure the word importance by attributing the NMT output to every input word through a gradient-based method. We validate the approach on a couple of perturbation operations, language pairs, and model architectures, demonstrating its superiority on identifying input words with higher influence on translation performance. Encouragingly, the calculated importance can serve as indicators of input words that are under-translated by NMT models. Furthermore, our analysis reveals that words of certain syntactic categories have higher importance, while the categories vary across language pairs, which can inspire better design princi-

ples of NMT architectures for multi-lingual translation. The chapter is organized as follows: we first introduce the problem background in §5.1 and present our method in §5.2. We then confirm the effectiveness of our method under various settings in §5.3. The linguistic analysis is demonstrated in §5.4. We then discuss some potential applications in §5.5 and conclude this chapter in §5.6.

5.1 Problems and Motivation

Neural machine translation (NMT) has achieved the state-of-the-art results on a mass of language pairs with varying structural differences, such as English-French [15, 148] and Chinese-English [66]. However, so far not much is known about how and why NMT works, which pose great challenges for debugging NMT models and designing optimal architectures.

The understanding of NMT models has been approached primarily from two complementary perspectives. The first thread of work aims to understand the importance of representations by analyzing the linguistic information embedded in representation vectors [131, 18] or hidden units [17, 42]. Another direction focuses on understanding the importance of input words by interpreting the input-output behavior of NMT models. Previous work [8] treats NMT models as black-boxes and provides explanations that closely resemble the attention scores in NMT models. However, recent studies reveal that attention does not provide meaningful explanations since the relationship between attention scores and model output is unclear [78].

In this chapter, we focus on the second thread and try to open the black-box by exploiting the gradients in NMT generation,

which aims to estimate the word importance better. Specifically, we employ the *integrated gradients* method [140] to attribute the output to the input words with the integration of first-order derivatives. We justify the gradient-based approach via quantitative comparison with black-box methods on a couple of perturbation operations, several language pairs, and two representative model architectures, demonstrating its superiority on estimating word importance.

We analyze the linguistic behaviors of words with the importance and show its potential to improve NMT models. First, we leverage the word importance to identify input words that are under-translated by NMT models. Experimental results show that the gradient-based approach outperforms both the best black-box method and other comparative methods. Second, we analyze the linguistic roles of identified important words, and find that words of certain syntactic categories have higher importance while the categories vary across language. For example, nouns are more important for Chinese⇒English translation, while prepositions are more important for English-French and -Japanese translation. This finding can inspire better design principles of NMT architectures for different language pairs. For instance, a better architecture for a given language pair should consider its own language characteristics.

5.2 Methodology

5.2.1 Word Importance

In this chapter, the notion of "word importance" is employed to quantify the contribution that a word in the input sentence makes to the NMT generations. We categorize the methods of

word importance estimation into two types: *black-box* methods without the knowledge of the model and *white-box* methods that have access to the model internal information (e.g., parameters and gradients). Previous studies mostly fall into the former type, and in this study, we investigate several representative black-box methods:

- *Content Words*: In linguistics, all words can be categorized as either content or content-free words. Content words consist mostly of nouns, verbs, and adjectives, which carry descriptive meanings of the sentence and thereby are often considered as important.
- *Frequent Words*: We rank the relative importance of input words according to their frequency in the training corpus. We do not consider the top 50 most frequent words since they are mostly punctuation and stop words.
- *Causal Model* [8]: Since the causal model is complicated to implement and its scores closely resemble attention scores in NMT models. In this study, we use *Attention* scores to simulate the causal model.

Our approach belongs to the white-box category by exploiting the intermediate gradients, which will be described in the next section.

5.2.2 Integrated Gradients

In this chapter, we resort to a gradient-based method, integrated gradients [140] (IG), which was originally proposed to attribute the model predictions to input features. It exploits the handy model gradient information by integrating first-order

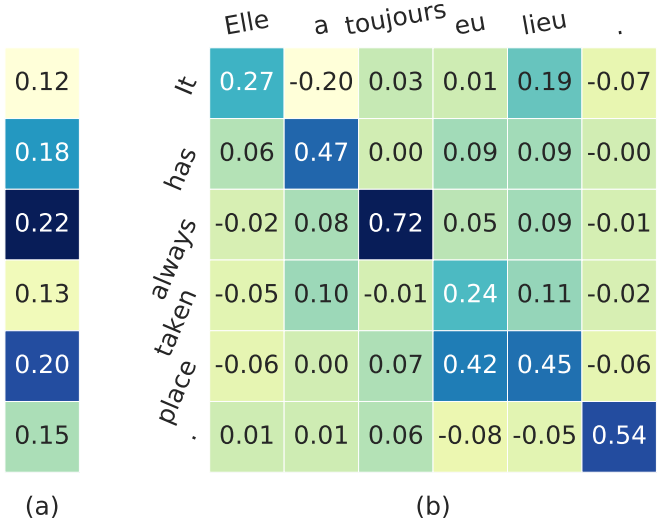


Figure 5.1: An example of (a) word importance and (b) contribution matrix calculated by integrated gradients on English⇒French translation task.

derivatives. IG is implementation invariant and does not require neural models to be differentiable or smooth, thereby is suitable for complex neural networks like Transformer. In this chapter, we use IG to estimate the word importance in an input sentence precisely.

Formally, let $\mathbf{x} = (x_1, ..., x_I)$ be the input sentence and $\mathbf{x}'$ be a baseline input. F is a well-trained NMT model, and $F(\mathbf{x})_j$ is the model output (i.e., $P(y_j|\mathbf{y}_{<j}, \mathbf{x})$) at time step j. Integrated gradients is then defined as the integral of gradients along a straight line path from the baseline $\mathbf{x}'$ to the input $\mathbf{x}$. In detail, the contribution of the i^{th} word in $\mathbf{x}$ to the prediction of $F(\mathbf{x})_j$ is defined as follows.

$$IG_i^j(\mathbf{x}) = (\mathbf{x}_i - \mathbf{x}'_i) \int_{\alpha=0}^{1} \frac{\partial F(\mathbf{x}' + \alpha(\mathbf{x} - \mathbf{x}'))_j}{\partial \mathbf{x}_i} \mathrm{d}\alpha$$

where $\frac{\partial F(\mathbf{x})_j}{\partial \mathbf{x}_i}$ is the gradient of $F(\mathbf{x})_j$ w.r.t. the embedding of the i^{th} word. In this chapter, as suggested, the baseline input $\mathbf{x}'$ is set as a sequence of zero embeddings that has the same sequence

length I. In this way, we can compute the contribution of a specific input word to a designated output word. Since the above formula is intractable for deep neural models, we approximate it by summing the gradients along a multi-step path from baseline $\mathbf{x}'$ to the input $\mathbf{x}$.

$$IG_i^j(\mathbf{x}) = \frac{(\mathbf{x}_i - \mathbf{x}_i')}{S} \sum_{k=0}^{S} \frac{\partial F(\mathbf{x}' + \frac{k}{S}(\mathbf{x} - \mathbf{x}'))_j}{\partial \mathbf{x}_i}$$

where S denotes the number of steps that are uniformly distributed along the path. The IG will be more accurate if a larger S is used. In our preliminary experiments, we varied the steps and found 300 steps yielding fairly good performance.
Following the formula, we can calculate the contribution of every input word makes to every output word, forming a contribution matrix of size $I \times J$, where J is the output sentence length. Given the contribution matrix, we can obtain the *word importance* of each input word to the entire output sentence. To this end, for each input word, we first aggregate its contribution values to all output words by the *sum* operation, and then normalize all sums through the *Softmax* function. Figure 5.1 illustrates an example of the calculated word importance and the contribution matrix, where an English sentence is translated into a French sentence using the Transformer model. The input in English is "It has always taken place ." and the output in French is "Elle a toujours eu lieu .". A negative contribution value indicates that the input word has negative effects on the output word.

5.3 Evaluations

Data To make the conclusion convincing, we first choose two large-scale datasets that are publicly available, i.e., Chinese-

English and English-French. Since English, French, and Chinese all belong to the subject-verb-object (SVO) family, we choose another very different subject-object-verb (SOV) language, Japanese, which might bring some interesting linguistic behaviors in English-Japanese translation.
For Chinese-English task, we use WMT17 Chinese-English dataset that consists of 20.6M sentence pairs. For English-French task, we use WMT14 English-French dataset that comprises 35.5M sentence pairs. For English-Japanese task, we follow [104] to use the first two sections of WAT17 English-Japanese dataset that consists of 1.9M sentence pairs. Following the standard NMT procedure, we adopt the standard byte pair encoding (BPE) [129] with 32K merge operations for all language pairs. We believe that these datasets are large enough to confirm the rationality and validity of our experimental analyses.

Implementation We choose the state-of-the-art model, Transformer [148] and the conventional RNNSearch model [15] as our test bed. We implement the *Attribution* method based on the Fairseq framework for the above models. All models are trained on the training corpus for 100k steps under the standard settings, which achieve comparable translation results. All the following experiments are conducted on the test dataset, and we estimate the input word importance using the model generated hypotheses.
In the following experiments, we compare IG (*Attribution*) with several black-box methods (i.e., *Content*, *Frequency*, *Attention*) as introduced in Section 5.2.1. In Section 5.3.1, to ensure that the translation performance decrease attributes to the selected

words instead of the perturbation operations, we randomly select the same number of words to perturb (*Random*), which serves as a baseline. Since there is no ranking for content words, we randomly select a set of content words as important words. To avoid the potential bias introduced by randomness (i.e., *Random* and *Content*), we repeat the experiments for 10 times and report the averaged results. We calculate the *Attention* importance in a similar manner as the *Attribution*, except that the attention scores use a *max* operation due to the better performance.

Evaluation We evaluate the effectiveness of estimating word importance by the translation performance decrease. More specifically, unlike the usual way, we measure the decrease of translation performance when perturbing a set of important words that are of top-most word importance in a sentence. The more translation performance degrades, the more important the word is.
We use the standard BLEU score as the evaluation metric for translation performance. To make the conclusion more convincing, we conduct experiments on different types of synthetic perturbations (Section 5.3.1), as well as different NMT architectures and language pairs (Section 5.3.2). In addition, we compare with a supervised erasure method, which requires ground-truth translations for scoring word importance (Section 5.3.3).

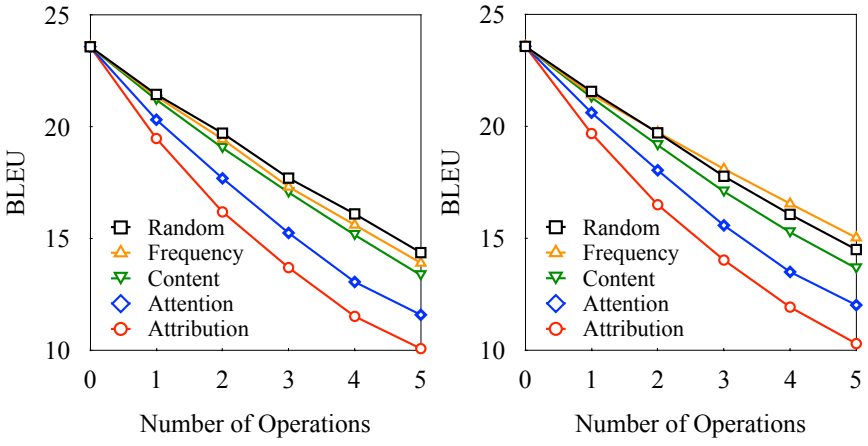
5.3.1 Results on Different Perturbations

In this experiment, we investigate the effectiveness of word importance estimation methods under different synthetic perturbations. Since the perturbation on text is notoriously hard [166] due to the semantic shifting problem, in this experiment, we

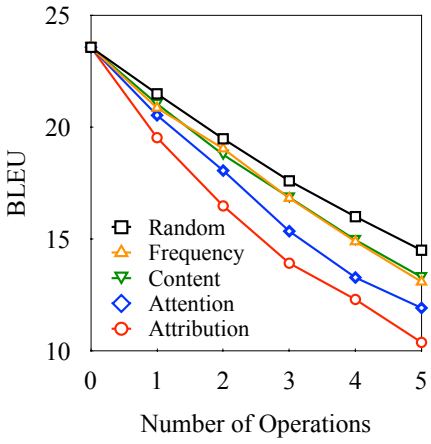
investigate three types of perturbations to avoid the potential bias :

- *Deletion* perturbation removes the selected words from the input sentence, and it can be regarded as a specific instantiation of sentence compression [33].
- *Mask* perturbation replaces embedding vectors of the selected words with all-zero vectors [10], which is similar to *Deletion* perturbation except that it retains the placeholder.
- *Grammatical Replacement* perturbation replaces a word by another word of the same linguistic role (i.e., POS tags), yielding a sentence that is grammatically correct but semantically nonsensical [30, 61], such as "*colorless green ideas sleep furiously*".

Figure 5.2 illustrates the experiments on Chinese⇒English translation with Transformer. It shows that *Attribution* method consistently outperforms other methods against different perturbations on a various number of operations. Here the operation number denotes the number of perturbed words in a sentence. Specifically, we can make the following observations.
Under three different perturbations, perturbing words of topmost importance leads to lower BLEU scores than *Random* selected words. It confirms the existence of important words, which have greater impacts on translation performance. Furthermore, perturbing important words identified by *Attribution* outperforms the *Random* method by a large margin (more than 4.0 BLEU under 5 operations).



(a) **Deletion** (b) **Mask**



(c) **Grammatical Replacement**

Figure 5.2: Effect of different perturbations on Chinese⇒English translation.

Finding 1: Important words are more influential on translation performance than the others.

Figure 5.2 shows that two *black-box* methods (i.e., *Content*, *Frequency*) perform only slightly better than the *Random* method. Specifically, the *Frequency* method demonstrates even worse performances under the *Mask* perturbation. Therefore, linguistic properties (such as POS tags) and the word frequency can only partially help identify the important words, but it is not as accurate as we thought. In the meanwhile, it is intriguing to explore what exact linguistic characteristics these important words reveal, which will be introduced in Section 5.4.

Finding 2: The gradient-based method is superior to comparative methods (e.g., Attention) in estimating word importance.

We also evaluate the *Attention* method, which bases on the encoder-decoder attention scores at the last layer of Transformer. Note that the *Attention* method is also used to simulate the best *black-box* method SOCRAT, and the results show that it is more effective than *black-box* methods and the *Random* baseline. Given the powerful *Attention* method, *Attribution* method still achieves best performances under all three perturbations. Furthermore, we find that the gap between *Attribution* and *Attention* is notably large (around 1.0+ BLEU difference). *Attention* method does not provide as accurate word importance as the *Attribution*, which exhibits the superiority of gradient-based methods and consists with the conclusion reported in the previous study [78].

In addition, as shown in Figure 5.2, the perturbation effectiveness of *Deletion*, *Mask*, and *Grammatical Replacement* varies

from strong to weak. In the following experiments, we choose *Mask* as the representative perturbation operation for its moderate perturbation performance, based on which we compare two most effective methods *Attribution* and *Attention*.

5.3.2 Results on Different Configurations

In this section, we mainly investigate whether our method can be generally applicable in various settings. Specifically, we vary the NMT architecture, language pair as well as direction to validate our method, and the details are presented as follows.

Different NMT Architecture We validate the effectiveness of the proposed approach using a different NMT architecture RNNSearch on the Chinese⇒English translation task. The results are shown in Figure 5.3(a). We observe that the *Attribution* method still outperforms both *Attention* method and *Random* method by a decent margin. By comparing to Transformer, the results also reveal that the RNNSearch model is less robust to these perturbations. To be specific, under the setting of five operations and *Attribution* method, Transformer shows a relative decrease of 55% on BLEU scores while the decline of RNNSearch model is 64%.

Different Language Pairs and Directions We further conduct experiments on another two language pairs (i.e., English⇒French, English⇒Japanese in Figures 5.3(b, c)) as well as the reverse directions (Figures 5.3(d, e, f)) using Transformer under the *Mask* perturbation. In all the cases, *Attribution* shows the best performance while *Random* achieves the worst result. More

specifically, *Attribution* method shows similar translation quality degradation on all three language-pairs, which declines to around the half of the original BLEU score with five operations.

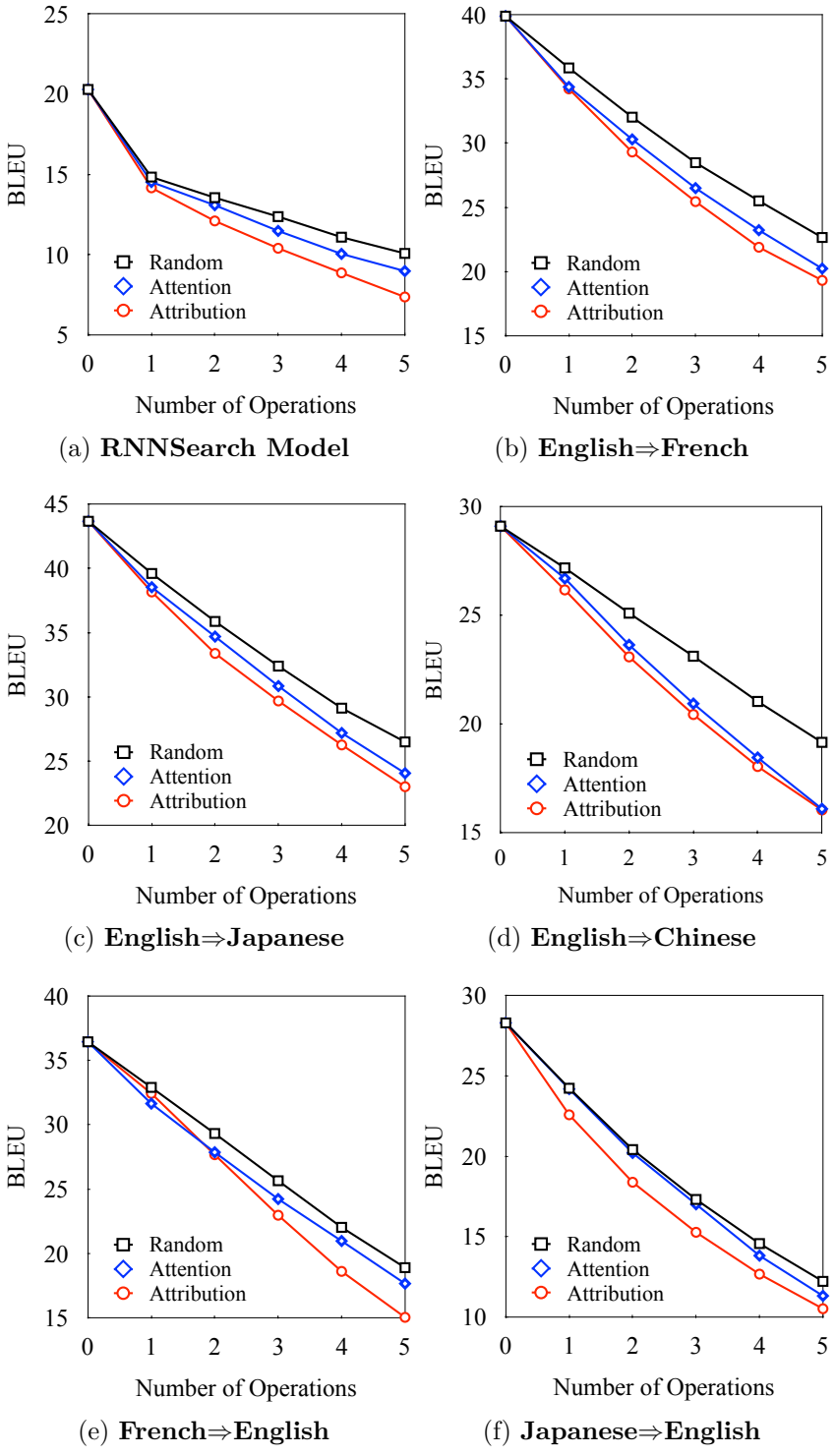
Finding 3: Our gradient-based method is consistent against different architectures, language pairs and translation directions

5.3.3 Comparison with Supervised Erasure

There exists another straightforward method, *Erasure* [8, 10, 170], which directly evaluates the word importance by measuring the translation performance degradation of each word. Specifically, it erases (i.e., *Mask*) one word from the input sentence each time and uses the BLEU score changes to denote the word importance (after normalization).

In Figure 5.4, we compare *Erasure* method with *Attribution* method under the *Mask* perturbation. The results show that *Attribution* method is less effective than *Erasure* method when only one word is perturbed. But it outperforms the *Erasure* method when perturbing 2 or more words. The results reveal that the importance calculated by erasing only one word cannot be generalized to multiple-words scenarios very well. Besides, the *Erasure* method is a supervised method which requires ground-truth references, and finding a better words combination is computation infeasible when erasing multiple words.

We close this section by pointing out that our gradient-based method consistently outperforms its black-box counterparts in various settings, demonstrating the effectiveness and universality of exploiting gradients for estimating word importance. In addition, our approach is on par with or even outperforms the



(a) **RNNSearch Model** (b) **English⇒French**

(c) **English⇒Japanese** (d) **English⇒Chinese**

(e) **French⇒English** (f) **Japanese⇒English**

Figure 5.3: Effect of the *Mask* perturbation on (a) Chinese⇒English translation using the RNNSearch model, (b, c, d, e, f) other language pairs and directions using Transformer model.

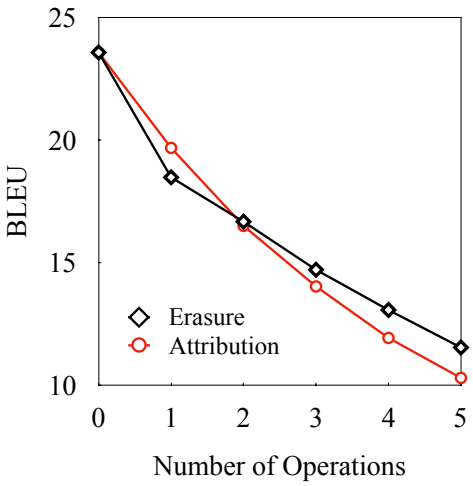


Figure 5.4: Effect of *Attribution* and *Erasure* methods on Chinese⇒English translation with *Mask* perturbation.

Method	Top 5%	Top 10%	Top 15%
Attention	0.058	0.077	0.119
Erasure	0.154	0.170	0.192
Attribution	**0.248**	**0.316**	**0.342**

Table 5.1: F1 accuracy of detecting under-translation errors with the estimated word importance.

supervised erasure method (on multiple-word perturbations). This is encouraging since our approach does not require any external resource and is fully unsupervised.

5.4 Analysis

In this section, we conduct analyses on two potential usages of word importance, which can help debug NMT models (Section 5.4.1) and design better architectures for specific languages (Section 5.4.2). Due to the space limitation, we only analyze the results of Chinese⇒English, English⇒French, and English⇒Japanese. We list the results on the reverse directions

in Appendix, in which the general conclusions also hold.

5.4.1 Effect on Detecting Translation Errors

In this experiment, we propose to use the estimated word importance to detect the under-translated words by NMT models. Intuitively, under-translated input words should contribute little to the NMT outputs, yielding much smaller word importance. Given 500 Chinese⇒English sentence pairs translated by the Transformer model (BLEU 23.57), we ask ten human annotators to manually label the under-translated input words, and at least two annotators label each input-hypothesis pair. These annotators have at least six years of English study experience, whose native language is Chinese. Among these sentences, 178 sentences have under-translation errors with 553 under-translated words in total.

Table 5.1 lists the accuracy of detecting under-translation errors by comparing words of *least* importance and human-annotated under-translated words. As seen, our *Attribution* method consistently and significantly outperforms both *Erasure* and *Attention* approaches. By exploiting the word importance calculated by *Attribution* method, we can identify the under-translation errors automatically without the involvement of human interpreters. Although the accuracy is not high, it is worth noting that our under-translation method is very simple and straightforward. This is potentially useful for debugging NMT models, e.g., automatic post-editing with constraint decoding [76, 120].

5.4.2 Analysis on Linguistic Properties

In this section, we analyze the linguistic characteristics of important words identified by the attribution-based approach. Specifically, we investigate several representative sets of linguistic properties, including POS tags, and fertility, and depth in a syntactic parse tree. Fertility can be categorized into 4 types: one-to-many ("≥ 2"), one-to-one ("1"), many-to-one ("$(0, 1)$"), and null-aligned ("0"). Syntactic depth shows the depth of a word in the dependency tree. In these analyses, we multiply the word importance with the corresponding sentence length for fair comparison. We use a decision tree based regression model to calculate the correlation between the importance and linguistic properties.

Table 5.2 lists the correlations, where a higher value indicates a stronger correlation. We find that the syntactic information is almost independent of the word importance value. Instead, the word importance strongly correlates with the POS tags and fertility features, and these features in total contribute over 95%. A lower tree depth indicates closer to the root node in the dependency tree, which might indicate a more important word. Therefore, in the following analyses, we mainly focus on the POS tags (Table 5.3) and fertility properties (Table 5.4). For better illustration, we calculate the distribution over the linguistic property based on both the *Attribution* importance ("Attri.") and the word frequency ("Count") inside a sentence, and the "$\triangle$" denotes relative change over the count-based distribution. The larger the relative increase between these two values, the more important the linguistic property is.

As shown in Table 5.3, *content* words are more important on Chinese$\Rightarrow$English but *content-free* words are more important

	Type	Zh⇒En	En⇒Fr	En⇒Ja
POS Tags	Noun	**21.0%**	1.9%	0.7%
	Verb	0.3%	**25.0%**	0.3%
	Adj.	0.4%	9.3%	0.7%
	Prep.	1.3%	4.5%	**26.7%**
	Dete.	3.0%	5.7%	2.1%
	Punc.	3.5%	**18.3%**	**30.5%**
	Others	0.5%	1.2%	4.7%
	Total	30.0%	65.9%	65.6%
Fertility	≥ 2	**50.2%**	**21.4%**	**21.7%**
	1	**15.4%**	7.0%	3.1%
	$(0, 1)$	2.5%	0.4%	3.0%
	0	0.0%	1.9%	3.8%
	Total	68.1%	30.7%	31.6%
Syntactic	Low	1.6%	2.5%	1.2%
	Middle	0.3%	0.8%	1.4%
	High	0.0%	0.1%	0.1%
	Total	1.9%	3.4%	2.7%

Table 5.2: Correlation between *Attribution* word importance with POS tags, fertility, and syntactic depth.

on English⇒Japanese. On English⇒French, there is no notable increase or decrease of the distribution since English and French are in essence very similar. We also obtain some specific findings of great interest. For example, we find that noun is more important on Chinese⇒English translation, while preposition is more important on English⇒French translation. More interestingly, English⇒Japanese translation shows a substantial discrepancy in contrast to the other two language pairs. The results reveal that preposition and punctuation are very important in English⇒Japanese translation, which is counter-intuitive.

Finding 4: Certain syntactic categories have higher importance while the categories vary across language pairs.

Punctuation in NMT is understudied since it carries little information and often does not affect the understanding of a sentence. However, we find that punctuation is important on English⇒Japanese translation, whose proportion increases dramatically. We conjecture that it is because the punctuation could affect the sense groups in a sentence, which further benefits the syntactic reordering in Japanese.

We further compare the fertility distribution based on word importance and the word frequency on three language pairs. We hypothesize that a source word that corresponds to multiple target words should be more important since it contributes more to both sentence length and BLEU score.

Table 5.4 lists the results. Overall speaking, one-to-many fertility is consistently more important on all three language pairs, which confirms our hypothesis. On the contrary, null-aligned words receive much less attention, which shows a persistently decrease on three language pairs. It is also reasonable since null-aligned input words contribute almost nothing to the translation outputs.

Finding 5: Words of high fertility are always important.

5.4.3 Analyses on Reverse Directions

We analyze the distribution of syntactic categories and word fertility on the same language pairs with reverse directions, i.e., English⇒Chinese, French⇒English, and Japanese⇒English. The results are shown in Table 5.5 and Table 5.6 respectively, where we observe similar findings as before. We use the Stanford POS tagger to parse the English and French input sentences, and use

Type		Chinese⇒English			English⇒French			English⇒Japanese		
		Count	Attri.	Δ	Count	Attri.	Δ	Count	Attri.	Δ
Content	Noun	0.383	0.407	*+6.27%*	0.341	0.355	*+4.11%*	0.365	0.336	*-7.95%*
	Verb	0.165	0.160	*-3.03%*	0.146	0.131	*-10.27%*	0.127	0.123	*-3.15%*
	Adj.	0.032	0.029	*-9.38%*	0.076	0.072	*-5.26%*	0.094	0.088	*-6.38%*
	Total	0.579	0.595	*+2.76%*	0.563	0.558	*-0.89%*	0.587	0.547	*-6.81%*
Content-Free	Prep.	0.056	0.051	*-8.93%*	0.120	0.132	*+10.00%*	0.129	0.151	*+17.05%*
	Dete.	0.043	0.043	*0.00%*	0.102	0.101	*-0.98%*	0.112	0.103	*-8.04%*
	Punc.	0.137	0.131	*-4.38%*	0.100	0.091	*-9.00%*	0.096	0.120	*+25.47%*
	Others	0.186	0.179	*-3.76%*	0.115	0.118	*+2.61%*	0.076	0.079	*+3.95%*
	Total	0.421	0.405	*-3.80%*	0.437	0.442	*+1.14%*	0.413	0.453	*+9.69%*

Table 5.3: Distribution of syntactic categories based on word count ("Count") and *Attribution* importance ("Attri.").

Languages	**Fertility**	Count	Attri.	△
Zh⇒En	≥ 2	0.087	0.146	*+67.82%*
	1	0.621	0.622	*+0.16%*
	$(0, 1)$	0.115	0.081	*-29.57%*
	0	0.176	0.150	*-14.77%*
En⇒Fr	≥ 2	0.126	0.138	*+9.52%*
	1	0.672	0.670	*-0.30%*
	$(0, 1)$	0.116	0.113	*-2.59%*
	0	0.086	0.079	*-8.14%*
En⇒Ja	≥ 2	0.117	0.143	*+22.22%*
	1	0.570	0.565	*-0.88%*
	$(0, 1)$	0.059	0.055	*-6.78%*
	0	0.254	0.237	*-6.69%*

Table 5.4: Distributions of word fertility and their relative change based on *Attribution* importance and word count.

the Kytea[1] to parse the Japanese input sentences.

Syntactic Categories On English⇒Chinese, *content* words are more important than *content-free* words, while the situation is reversed on both French⇒English and Japanese⇒English translations. Since there is no clear boundary between Preposition/Determiner and other categories in Japanese, we set both categories to be none. Similarly, Punctuation is more important on Japanese⇒English, which is in line with the finding on English⇒Japanese. Overall speaking, it might indicate that the Syntactic distribution with word importance is language-pair related instead of the direction.

Word Fertility The word fertility also shows similar trend as the previously reported results, where one-to-many fertility is more important and null-aligned fertility is less important.

[1] http://www.phontron.com/kytea/

Type		English⇒Chinese			French⇒English			Japanese⇒English		
		Count	Attri.	Δ	Count	Attri.	Δ	Count	Attri.	Δ
Content	Noun	0.313	0.338	*+7.99%*	0.323	0.313	*-3.10%*	0.426	0.377	*-11.50%*
	Verb	0.132	0.127	*-3.79%*	0.172	0.160	*-6.98%*	0.091	0.085	*-6.59%*
	Adj.	0.091	0.094	*+3.30%*	0.078	0.077	*-1.28%*	0.014	0.012	*-14.29%*
	Total	0.536	0.559	*+4.29%*	0.572	0.551	*-3.67%*	0.531	0.473	*-10.92%*
Content-Free	Prep.	0.133	0.129	*-3.01%*	0.116	0.125	*+7.76%*	-	-	-
	Dete.	0.122	0.113	*-7.38%*	0.123	0.126	*+2.44%*	-	-	-
	Punc.	0.088	0.078	*-11.36%*	0.076	0.084	*+10.53%*	0.091	0.122	*+34.07%*
	Others	0.121	0.121	*0.00%*	0.113	0.114	*+0.88%*	0.377	0.405	*+7.43%*
	Total	0.464	0.441	*-4.96%*	0.428	0.449	*+4.91%*	0.469	0.527	*+12.37%*

Table 5.5: Distribution of syntactic categories with reverse directions based on word count ("Count") and *Attribution* importance ("Attri.").

Languages	**Fertility**	Count	Attri.	△
En⇒Zh	≥ 2	0.091	0.106	*+16.48%*
	1	0.616	0.629	*+2.11%*
	$(0, 1)$	0.083	0.077	*-7.23%*
	0	0.210	0.187	*-10.95%*
Fr⇒En	≥ 2	0.088	0.094	*+6.82%*
	1	0.707	0.721	*+1.98%*
	$(0, 1)$	0.102	0.094	*-7.84%*
	0	0.103	0.092	*-10.68%*
Ja⇒En	≥ 2	0.079	0.085	*+7.59%*
	1	0.513	0.520	*+1.36%*
	$(0, 1)$	0.086	0.097	*+12.79%*
	0	0.322	0.298	*-7.45%*

Table 5.6: Distributions of word fertility and relative changes with reverse directions.

Interestingly, many-to-one fertility shows an increasing trend on Japanese⇒English translation, but the proportion is relatively small.

In summary, the findings on language pairs with reverse directions still agree with the findings in this chapter, which further confirms the generality of our experimental findings.

5.5 Discussion

We approach understanding NMT by estimating the word importance via a gradient-based methods. Our analyses show that important words are of distinct syntactic categories on different language pairs, which might support the viewpoint that essential inductive bias should be introduced into the model design [138]. Our study also suggests the possibility of detecting the notorious under-translation problem via the gradient-based method.

This chapter presents an initiating step towards the general understanding of NMT models, which may bring some potential improvements, such as

- *Interactive MT and Constraint Decoding* [52, 76]: The model pays more attention to the detected unimportant words, which are possibly under-translated;
- *Adaptive Input Embedding* [14]: We can extend the adaptive softmax [60] to the input embedding of variable capacity – more important words are assigned with more capacity;
- *NMT Architecture Design*: The language-specific inductive bias (e.g., different behaviors on POS [108, 161]) should be incorporated into the model design.

We can also explore other applications of word importance to improve NMT models, such as more tailored training methods. In general, model interpretability can build trust in model predictions, help error diagnosis and facilitate model refinement. We expect our work could shed light on the NMT model understanding and benefit the model improvement.

There are many possible ways to implement the general idea of exploiting gradients for model interpretation. The aim of this chapter is not to explore this whole space but simply to show that some fairly straightforward implementations work well. Our approach can benefit from advanced exploitation of the gradients or other useful intermediate information, which we leave to the future work.

5.6 Summary

In this chapter, we propose to understand NMT by investigating the word importance via a gradient-based method, which bridges the gap between word importance and translation performance. Empirical results show that the gradient-based method is superior to several black-box methods in estimating the word importance. Our study demonstrates the necessity and effectiveness of exploiting the intermediate gradients for estimating word importance. We find that word importance is useful for understanding NMT by identifying under-translated words. We provide empirical support for the design principle of NMT architectures: essential inductive bias (e.g., language characteristics) should be considered for model design.

□ **End of chapter.**

Chapter 6

Phrase-table-based Bilingual Knowledge Assessment

In the previous chapter, we propose to interpret the neural machine translation model from the input-output attribution perspective. However, the input-output attribution focuses only on specific input examples, and it does not represent an global explanation of the intelligent software behaviors. Therefore, in this chapter, we attempt to bridge the gap by assessing the bilingual knowledge learned in NMT models with the *phrase table* – an interpretable table of bilingual lexicons. The proposed method provides a global explanation for the bilingual knowledge learned in the model. Extensive experiments on widely-used datasets show that the phrase table is reasonable and consistent against language pairs, random seeds, and model structures. Equipped with the interpretable phrase table, we find some interesting findings in the model training and recent advances for model improvement. The chapter is organized as follows: we first introduce the problem background in §6.1 and present our method in §6.2. We then confirm the effectiveness of our method under various settings in §6.3. With the proposed method, we analyze the training process and some recent

advances of NMT in §6.4. We then discussion the potential applications of our method in §6.5 and conclude this chapter in §6.6.

6.1 Problems and Motivation

Modern machine translation (MT) systems aim to produce fluent and adequate translations by automatically learning in-depth knowledge of bilingual lexicons and grammar from the training corpora. Since the proposal of machine translation task, techniques have evolved from rule-based MT (RBMT) [67, 126], through SMT [26, 110], to NMT [141, 15]. In RBMT methods, a large number of linguistic rules and extensive lexicons with morphological, syntactic, and semantic information, are manually constructed by humans. But the method is impractical in nowadays since the rules are very complex and are always evolving. Thanks to the availability of large amounts of parallel data in the wild, SMT approaches automatically learn the linguistic knowledge from bilingual corpora with statistic models, which relieve the labor-intensive problem of RBMT methods. More recently, NMT, which builds an end-to-end neural network on the training data, has taken the field of MT. Several studies have shown that NMT model representations contain a substantial amount of linguistic information on multiple levels: morphological [18], syntactic [131], and semantic [75].

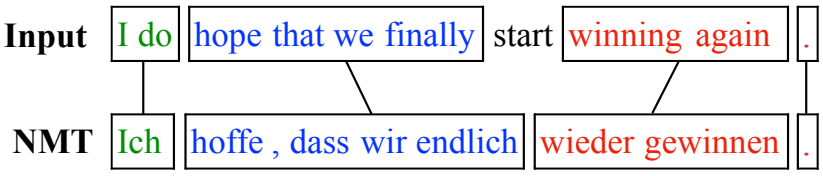
In the development circle of each generation, MT models are generally improved with techniques that are essential in the last generation. For example, Chiang et al. [29] and Liu et al. [92] relieved the non-fluent translation problem of SMT models by automatically learning syntactic rules that were manually

created in RBMT systems. Tu et al. [145] alleviated the inadequate translation problem of NMT models by introducing the coverage mechanism, which is a standard concept in SMT. Inspired by previous studies, we hypothesize that MT models of different generations are possibly identical in modeling the essential knowledge.

Specifically, SMT models generate translations based on several statistical models that *explicitly* represent the knowledge bases, such as translation model for bilingual lexicons, reordering, and language models for grammar [83]. Recently, NMT models have advanced the state-of-the-art (SOTA) by *implicitly* modeling the knowledge bases in a large neural network, which is jointly trained to improve the translation performance [15, 57, 149]. Despite their power with a massive amount of parameters, we have limited understanding of how and why an NMT model works, which poses great challenges for error analysis and model refinement.

In this part, we bridge the gap by assessing the knowledge bases learned by NMT models with statistical models in SMT. We believe and empirically verify that although using different forms (e.g., continuous vs. discrete) to represent the knowledge, NMT and SMT models are identical in modeling the essential knowledge bases. In the long-goal journey, we start with probing the *bilingual knowledge* with the statistical translation model, also known as the *phrase table*. Bilingual knowledge is at the core of adequacy modeling, which is a major weakness of NMT models [145].

The phrase table is an essential component of the SMT systems, which records the correspondence between bilingual lexicons [83]. Previous studies have incorporated the phrase



(a) Output of an English ⇒ German NMT model

Source	*Target*
I do	Ich
I do hope that	hoffe ich , dass
hope that we finally	hoffe , dass wir endlich
winning again	wieder gewinnen
winning again	gewinnen einer
.	.
...	

(b) Phrase table extracted from the NMT model

Figure 6.1: The output of an NMT model (a) can be explained by an extracted phrase table (b).

table as an external signal to guide the generation in NMT models [153, 164, 167, 62]. All these works show that the bilingual knowledge in phrase tables can be identical to those in NMT models, and thereby can be seamlessly integrated into NMT models. Based on the observation, we employ the phrase table to assess the bilingual knowledge for NMT models. In addition, Lample et al. [87] have advanced the SOTA performance of unsupervised NMT by evolving from learning the alignment of word embeddings to phrase embeddings based on an external phrase table. The improvement is identical to the evolution of SMT from word-based models [26] to phrase-based models [85]. It reconfirms our hypothesis that MT models of different generations are identical in modeling the essential knowledge, and thus share similar evolving trends.

Specifically, we extract the phrase table from NMT model predictions, which is inspired by recent work on investigating the example forgetting phenomenon in image classification [144]. Intuitively, an NMT model has learned the essential knowledge in terms of bilingual phrase pairs if it can correctly predict the corresponding part of a training example. Experimental results on different language pairs, random seeds, and model structures show that the extracted phrase tables correlate well with the NMT performances. Besides, as shown in Figure 6.1, the phrase table extracted from an NMT model can well explain the model translation. These results reveal that the phrase table can reasonably represent the bilingual knowledge learned by NMT models.

With the interpretable phrase table in hand, we could better understand the behaviors of NMT models in many aspects. We start with investigating the learning dynamics of the bilingual knowledge. We find that NMT models tend to first learn simple patterns and then complex patterns, and the catastrophic forgetting phenomenon occurs during the model training.[1] Besides, we reveal that one strength of NMT models over SMT models is that, NMT models distill high-quality bilingual knowledge from the training data. We then revisit several advances in improving NMT models, which potentially affect the learned bilingual knowledge. Through extensive experiments, we have the following observations:

- *Model Capacity*: We thought it likely that increasing model capacity leads to more bilingual lexicons. It turns out to be false: Transformer-Big outperforms Transformer-Base

[1] We follow [144] to define "forgetting event" to have occurred when a training example transitions from being predicted correctly to incorrectly during training.

by 1.3 BLEU points, while the extracted phrase tables are of almost the same size. We conjecture that the strengths of large-capacity models lie in a better learning ability of more complex knowledge, such as composition rules to combine the bilingual lexicons.

- *Data Augmentation*: We investigate back translation [129] and forward translation [165, 68], which introduce additionally synthetic parallel corpus. Both the back-translation [129] and the forward-translation [165] improve the translation performance not only by introducing new bilingual knowledge but also with a better quality estimation of existing knowledge.

- *Domain Adaptation*: Fine-tuning is a simple yet effective technique in domain adaptation, which learns to transfer out-of-domain knowledge to in-domain [96]. As expected, by adapting to the in-domain data, the fine-tuning approach learns more and better bilingual knowledge from the in-domain data while forgetting partial out-of-domain knowledge.

6.2 Methodology

There are many possible ways to implement the general idea of extracting the phrase table from NMT model predictions. The aim of this chapter is not to explore the whole space but simply to show that one fairly straightforward implementation works well and the idea is reasonable. We leave the exploitation of more advanced statistic models on bilingual knowledge (e.g., syntax rules [92] and discontinuous phrases [56]) for future work.

As shown in Algorithm 2, we follow the standard pipeline in SMT to construct the phrase table with a two-phase approach. The first phase, as the focus of this chapter, is *phrase extraction* where the bilingual phrase pairs are extracted from a word-aligned parallel data. Secondly, each phrase pair is assigned with several scores, which are estimated based on the occurrences of these phrases or their words in the same word-aligned training data. The key challenge then is how to incorporate the prior of NMT predictions into the SMT pipeline. In this study, we model the NMT priors as a mask sequence, which is integrated into the standard SMT pipeline as a constraint, as listed in Algorithm 2.

Algorithm 2: Constructing Phrase Table

Input : training example $(\mathbf{x}, \mathbf{y})$, alignment $\mathbf{a}$, mask $\mathbf{m}$
Output: phrase set $\mathcal{R}$

Procedure `PhraseTable()`
  EXTRACTION;
  ESTIMATION;
Procedure `Extraction()`
  $\widehat{\mathcal{R}} \leftarrow$ extract candidates from $\{(\mathbf{x}, \mathbf{y}), \mathbf{a}\}$;
  foreach $r \in \widehat{\mathcal{R}}$ **do** ▷ `priors of NMT predictions`
    if r *is consistent with* $\mathbf{m}$ **then**
      $\mathcal{R}$.append(r)
    end
  end
Procedure `Estimation()`
  STANDARD PROCEDURE;

Building *Masked* Word-Aligned Parallel Data. Inspired by [144], we define "*memorized phrase pair*" to be extracted from the associated (partial) training example, which is predicted correctly by the NMT model. To this end, we first decompose the sequence

generation of NMT into a series of classification tasks.
Given a model M and a training example ($\mathbf{x} = \{x_1, \ldots, x_I\}, \mathbf{y} = \{y_1, \ldots, y_J\}$), we use the model M to force-decode $\mathbf{x}$ to $\mathbf{y}$ and check whether the j-th token is correctly predicted:

$$m_j = \begin{cases} 1, & \text{if } y_j = \arg\max_{y'_j \in V} P(y'_j | \mathbf{y}_{<j}, \mathbf{x}) \\ 0, & \text{otherwise} \end{cases}$$

where $P(y'_j | \mathbf{y}_{<j}, \mathbf{x})$ is the model prediction probability at the step j (ranges from 1 to J) and V is the target vocabulary. A token y_j is predicted correctly if the model assigns the highest probability to it.
Intuitively, a token y_j with mask $m_j = 0$ denotes that the token is not correctly predicted by the model. Accordingly, any phrase pairs that contain the token y_j should not be extracted from the training example $(\mathbf{x}, \mathbf{y})$, since these phrase pairs are not fully learned by the NMT model. A lightweight implementation is to replace these tokens with a special symbol "$MASK$", and follow the standard phrase extraction procedure as in the SMT pipeline. Then we remove phrase pairs that contain the symbol "$MASK$" (lines 6-10 in Algorithm 2), and feed the pruned phrase pairs to the second phase of quality score estimation.

6.3 Evaluations

6.3.1 Experimental Setup

Data and Models We conduct experiments on the widely-used WMT2014 English⇒German (En⇒De) and the syntactically-distant WAT2017 English⇒Japanese (En⇒Ja) [106] datasets. We use 4-gram NIST BLEU score [115] as the evaluation metric.

For all datasets used in the experiments, we apply the BPE [129] with 32K merge operations.

SMT experiments We follow the standard configuration in the SMT pipeline and settings of Edinburgh's phrase-based system in WMT-2014 [44]. More specifically, we leverage the huge German monolingual corpus on the WMT website and the Japanese data in the parallel corpus to train a 5-gram language model for En⇒De and En⇒Ja translation respectively, and the language model is built on the KenLM [73][2] toolkit. The fast_align [45][3] is used to obtain the word alignment. We use the Moses [84] default system setting for both training and testing. For the training, we set maximum phrase length to 7 for phrase table extraction and adopt MERT [109] to tune feature weights of the Moses decoder. Note that all experiments are automated by following standard procedures and parameter settings. Following [80] to reduce the redundancy, we further remove phrase pairs that occur only once in the training data. In following experiments with the phrase table, we adopt inverse phrase translation probability, inverse lexical weighting, direct phrase translation probability, direct lexical weighting, phrase penalty, word penalty. language model score, and lexical reordering features.

NMT experiments We use the Fairseq [114] toolkit to implement the Transformer models [149]. We follow the standard parameter configuration to train all the models and the batch size is 32K (4096×8) tokens. The TRANSFORMER-

[2]https://kheafield.com/code/kenlm/
[3]https://github.com/clab/fast_align

BIG model is trained for 300K steps with 0.3 dropout. The TRANSFORMER-SMALL model is trained following the same setting of TRANSFORMER-BASE. We train the NMT models for 100K steps and save the checkpoint per epoch. To understand the dramatic increase of NMT performances in the first epoch, we save checkpoints per 200 steps and extract phrase tables from training examples that have seen so far only.

Environment and Implementation details All NMT models are trained on 8 V100 Nvidia GPUs using the single precision floating point. The SMT experiments (including phrase table extraction and SMT model training) are based on CPUs only. Our SMT experiments are conducted using a Linux machine with CPU of 80-cores Intel(R) Xeon(R) Gold 6133 @2.50GHz and memory of 250GB. The experiments are efficient since many of which can be executed in parallel. Force decoding an NMT model on the training corpus can be conduced in parallel with a large batch, which takes less than ten minutes only. For the bilingual knowledge extraction, it takes around 2 hours to extract the phrase table from the force-decoded data. In the analysis part, the SMT model training time varies with the phrase table size, which ranges from around 2 to 9 hours.

Evaluation Metrics To verify our claims, we propose several metrics to quantitatively evaluate the phrase table quality. If the metrics correlate well with the NMT performance, the phrase table is reasonable in representing the bilingual knowledge of NMT models. The metrics are as follows:
Phrase Table Size: As a straightforward metric, the size measures the number of distinct phrase pairs in a phrase table.

A larger phrase table size indicates more abundant bilingual knowledge.

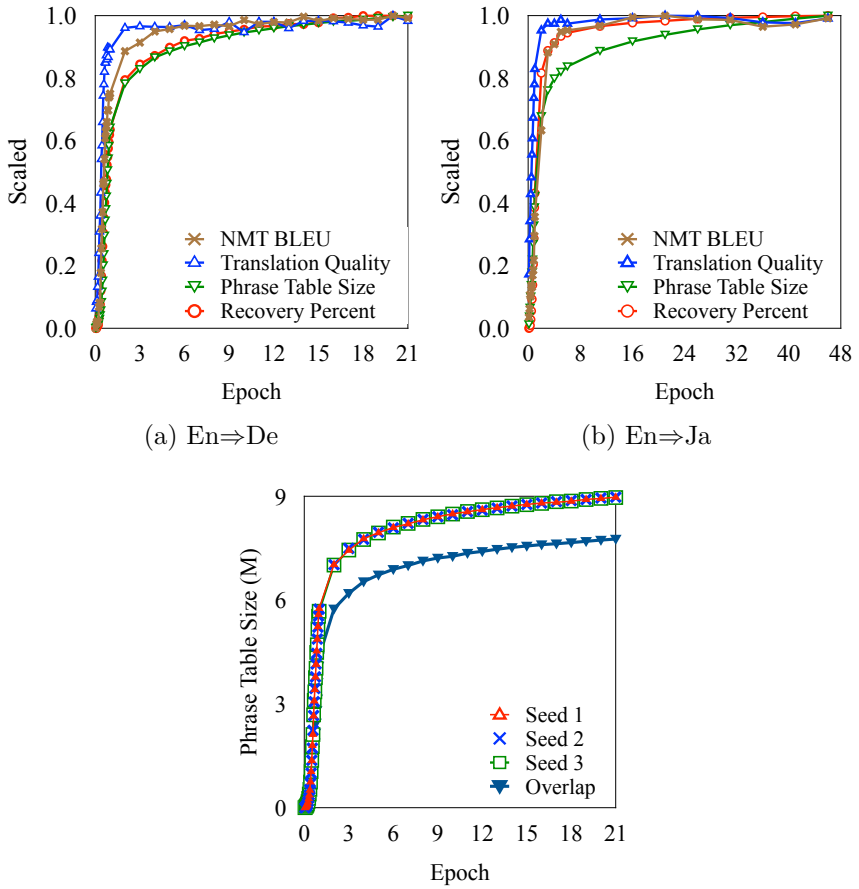
Recovery Percent: A good phrase table should contain enough knowledge to cover as much training data as possible. Accordingly, we propose the recovery percent metric to measure the ability of phrase tables in training data reconstruction. In detail, we use the phrase table to force decode the target sentence to recover as many target tokens as possible, and the proportion of the recovered tokens over all tokens is denoted as the recovery percent. A higher recovery percent indicates a better phrase table since more data can be reconstructed.

Translation Quality: Finally, we directly evaluate the phrase table quality in terms of translation performance. Specifically, we train a SMT model with the extracted phrase table by the off-the-shelf Moses toolkit and evaluate its BLEU score on the test set. For fair comparisons, we keep other SMT components unchanged and only alter the phrase table, therefore the relative SMT BLEU score is our focus of interest.

6.3.2 Phrase Table Evaluation

The extracted phrase table correlates well with the NMT performance. A good knowledge representation should be highly in line with the NMT performance during the entire learning process. Figure 6.2a illustrates the results of the above metrics on the En⇒De dataset. We evaluate the correlation between quality metrics of phrase table (i.e., phrase table size, recovery percent, and translation quality) and NMT performance ("NMT BLEU") on En⇒De and En⇒Ja datasets. All metrics are scaled by the corresponding best score to fit in the figure. As seen, all three metrics are highly in line with the NMT

performance ("NMT BLEU") in the learning course. The Pearson correlations between NMT BLEU scores and phrase table size, recovery percent, and the translation quality are 0.975, 0.987, and 0.956, respectively, demonstrating very high correlations between the phrase table and NMT performance. It confirms our claim that the phrase table is a reasonable assessment to represent the bilingual knowledge learned by NMT models.



(a) En⇒De (b) En⇒Ja

(c) En⇒De with Different Seeds

Figure 6.2: Correlation between phrase table quality and NMT performance on (a) En⇒De and (b) En⇒Ja and the phrase table size under different random seeds (c).

The conclusion is robust across language pairs, random seeds and model structures. We also validate our approach on the En⇒Ja dataset, as shown in Figure 6.2b. The Pearson correlations are respectively 0.988, 0.990, and 0.908, which demonstrate the universality of our conclusions. To avoid the potential bias, we vary the initialization seed and analyze the robustness of extracted phrase tables. Figure 6.2c depicts that the phrase table size increases similarly in different seeds. Besides, at each epoch, more than 85% phrase pairs are the same among three seeds ("Overlap"), showing its robustness against random seeds. Experiments on a LSTM-based model further confirm our findings and we leave the results in the Appendix due to space limits. Considering the general applicability of the phrase table, we use the Transformer model on En⇒De translation for further analyses. We will interchangeably use the terms "phrase table" and "bilingual knowledge" in the following sections.

6.3.3 Different Model Structures

In addition to the experiments on different language pairs and random seeds , we further evaluate the correlation between the quality metric of phrase tables and NMT performances ("NMT BLEU") on a state-of-the-art but different structure, a LSTM-based Transformer [43]. The results are depicted in the Figure 6.3. The Pearson correlation between NMT BLEU scores and phrase table size, recovery percent, and the translation quality are 0.990, 0.993, 0.900, respectively. The high correlation score demonstrate that the phrase table is robust against different model structures. The results reconfirm that the proposed phrase table is a reasonable assessment tool for representing the bilingual knowledge learned by NMT models.

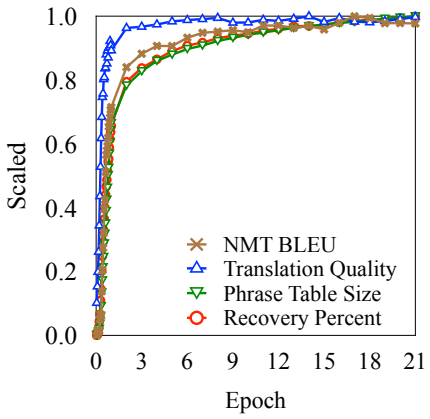


Figure 6.3: Correlations between phrase table quality and NMT performance for a LSTM-based model.

6.4 Analysis

With the interpretable phrase table in hand, we attempt to understand how NMT models learn the bilingual knowledge from two perspectives:

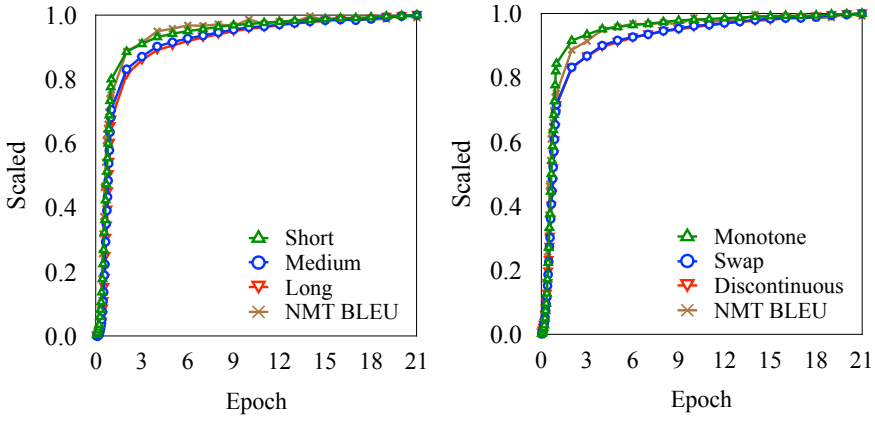
- How do NMT models learn the bilingual knowledge during training? (Section 6.4.1 and Section 6.4.2)
- Does the trained NMT model sufficiently exploit the bilingual knowledge embedded in the training examples? (Section 6.4.3)

6.4.1 Learning Dynamics

In this section, we investigate the evolvement of bilingual knowledge during the NMT model training. To this end, we categorize the phrase pair into different levels (in ascending order of complexity) using several metrics that are widely used in SMT research. For fair comparisons, the metric scores are normalized by the max score in each level.

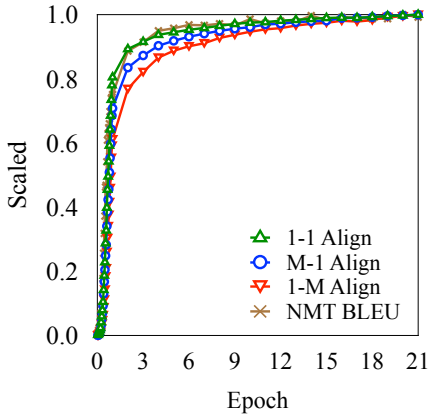
Phrase Length: A long phrase is usually difficult to translate, and thereby more complex than a short phrase [94]. The phrase length category is: *short* (1-3) < *middle* (3-5) < *long* (5-7).
Reordering Type: The metric measures the order of two phrases with lexicalized reordering [143]. Disordered phrases are often hard to translate [83] and thereby more complex. The category of reordering type is: *monotone* < *swap* < *discontinuous*.
Word Fertility: Word fertility measures the source-target word alignments in each phrase pair. Words with a complex fertility might indicate inherent translation difficulty [24]. Fertility type with increasing complexity is: *1-1 align* < *M-1 align* < *1-M align*.

NMT models tend to learn simple patterns first and complex patterns later. As shown in Figure 6.4a, NMT models learn short phrases faster than medium phrases and long phrases, embodied by the fastest convergence rate and the highest slope among three categories in the first epoch. As the learning continues, medium and long phrases start to converge to a relatively stable state slowly. Besides, NMT BLEU scores show a very similar increasing trend as the short phrase, demonstrating a high correlation (Pearson correlation: 0.992) between the NMT performance and short phrases.
We can observe similar findings on the phrase reordering type (Figure 6.4b) and word fertility (Figure 6.4c). Simple patterns like monotone and 1-1 aligned phrases can be quickly learned by NMT models, while complex patterns are learned more slowly. It is in line with the findings of [122]: deep networks will first learn low-complexity functional components, before absorbing high-complexity features. The results also indicate that NMT



(a) Phrase Length

(b) Reordering Type



(c) Word Fertility

Figure 6.4: Learning dynamics of bilingual knowledge according to three metrics of different complexity levels.

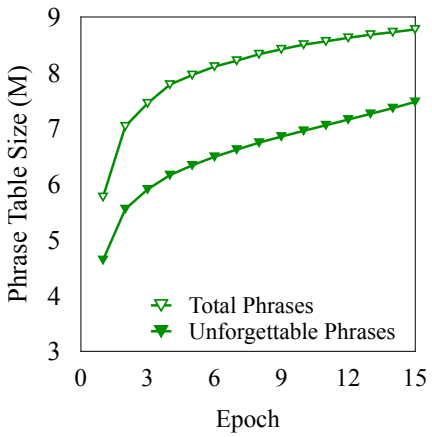
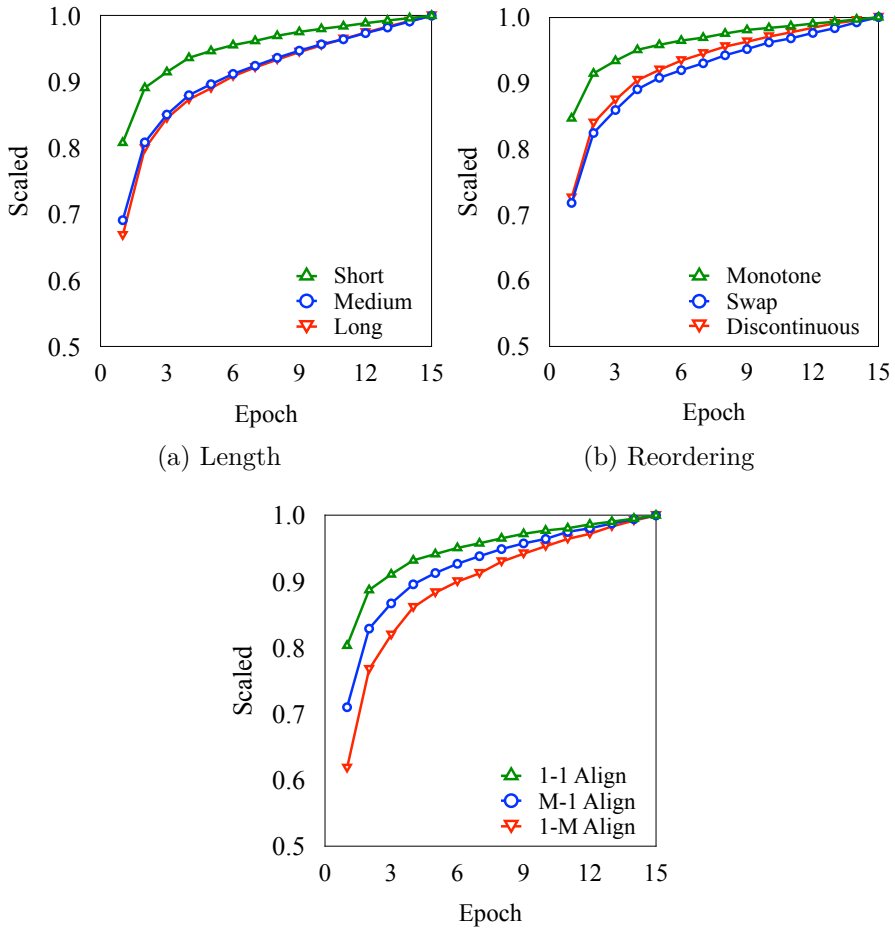


Figure 6.5: Learning dynamics of the total learned phrases and unforgettable phrases.

models might by nature has the learning ability similar to the *curriculum learning* [19, 82] without any explicit curriculum.

Forgetting dynamics occur in the learning of bilingual knowledge. As shown in Figures 6.2 and 6.4, the size of the learned phrase table is monotonically increasing as the learning processes. One question naturally arises: *are the phrase pairs never forgotten once learned?*

Figure 6.5 shows the result. Note that we only plot the first 15 epochs to ensure that the phrases are never forgotten until the training ends for at least several epochs. Around 80% of learned phrase table is unforgettable phrases (always learned phrase pairs), while the rest phrase pairs are forgotten. The finding is consistent with the findings of [144] on the image classification tasks. Further analyses are shown in the Appendix.

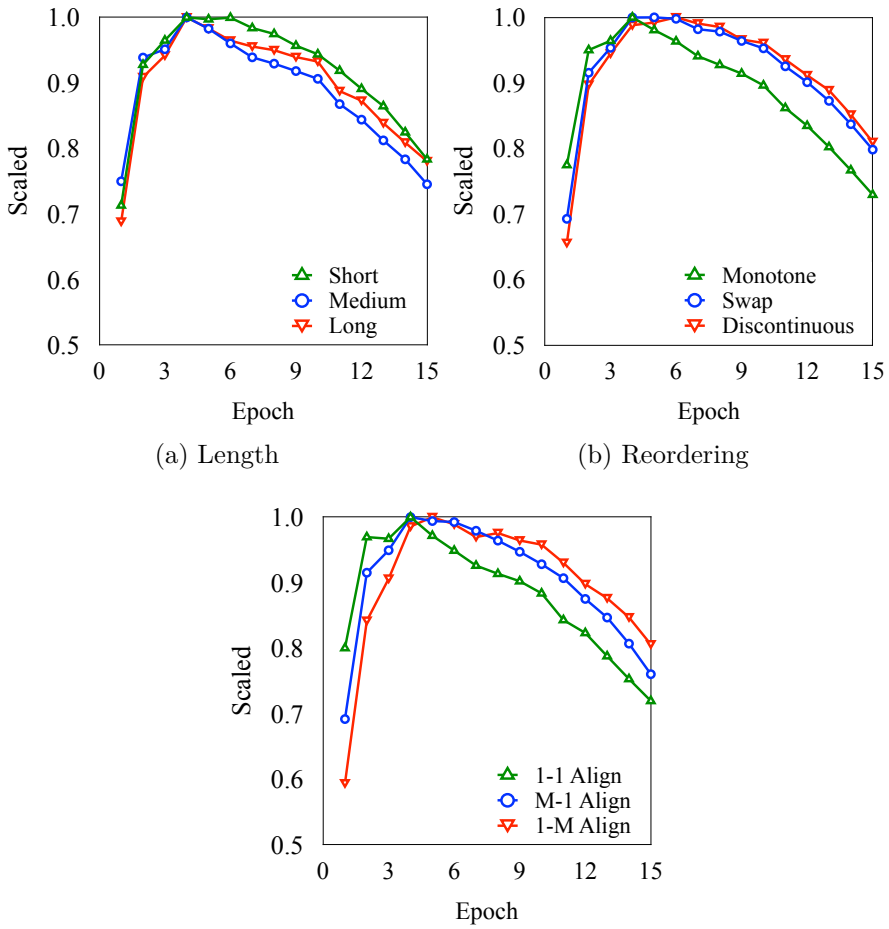


(a) Length

(b) Reordering

(c) Fertility

Figure 6.6: Learning dynamics of unforgettable bilingual knowledge according to different complexity metrics.



(a) Length

(b) Reordering

(c) Fertility

Figure 6.7: Learning dynamics of bilingual knowledge that are forgotten according to different complexity metrics.

6.4.2 Forgettable and Unforgettable Knowledge

In Section 6.4, we find that the model forgets some bilingual knowledge in the learning course. In each epoch, the bilingual knowledge learned by NMT models consists of two parts: unforgettable (i.e., never forgotten in subsequent epochs) and forgettable (i.e., the remaining part) knowledge. In this section, we further analyze the evolvement of both unforgettable and forgettable knowledge in the learning course in terms of metrics (i.e., phrase length, reordering type, and word fertility) of different complexity levels. Figure 6.6 and Figure 6.7 show the results of unforgettable and forgettable knowledge correspondingly and the scores are normalized by the max value in each metric.

For the unforgettable knowledge (Figure 6.6), since the very beginning epochs, the model has captured most of the simple and unforgettable patterns, as indicated by a high start point (around 80%). However, complex patterns (e.g., longer phrase length) although have a lower starting point, are quickly learned by the model. Besides, from the distribution of phrase pairs, we find that simple patterns occupy the majority of the unforgettable knowledge (Over 85%).

For the forgettable knowledge (Figure 6.7), the patterns in different complexity levels show a similar trend: they first increase to the peak and then decrease. The model continually forgets certain knowledge at the first several epochs, which might indicate a very unstable model training at the beginning. After that, the model gradually grasps more and more knowledge and the forgettable knowledge starts to decrease, potentially showing that the training becomes stabilized.

6.4.3 Learned Bilingual Knowledge

In this experiment, we evaluate whether NMT models have sufficiently exploited the bilingual knowledge in the training examples, by comparing the phrase tables extracted from NMT predictions and from the raw training data. We use the latter to represent the full bilingual knowledge embedded in all raw training examples.

"Shared" vs. "Non-Shared" Knowledge To compare the bilingual knowledge of two models, we first define the "Shared" knowledge as entries having same phrase pairs in both phrase tables. An entry in the phrase table also includes conditional probabilities and count. The "Shared" knowledge could still be different regarding the conditional probabilities and count even they have same phrase pairs. After finding the "Shared" knowledge, the complementary parts in each phrase table is denoted as the "Non-Shared" knowledge. In other words, "All" denotes the whole phrase table, "Shared" denotes the intersection of two tables, and "Non-shared" denotes the complement. Note that the probabilities of "Shared" phrases are different for the two tables.

As shown in Table 6.1, the bilingual knowledge learned by NMT model ("NMT") shows comparable translation quality with the full-data knowledge ("Full") (17.90 vs. 17.91), but with only *a half of* phrases (9.0M vs. 17.5M).[4] In addition, NMT provides a better probability estimation for the "Shared" phrase pairs (17.90 vs. 17.32), in other words, the distilled essential knowledge. In the "Non-Shared" phrase table, 78.2% of the

[4]Considering the full phrase table before filtering, NMT phrase only takes 22.8% of the full table (76M vs. 335M).

Phrase Table	Shared		Non-Shared		All	
	Size	*BLEU*	*Size*	*BLEU*	*Size*	*BLEU*
Full	9.0M	17.32	8.5M	4.50	17.5M	17.91
NMT	9.0M	17.90	0M	0	9.0M	17.90

Table 6.1: Comparison of the phrase table extracted from the full training data ("Full") and NMT models ("NMT").

Model	NMT		Phrase Table	
	#Para	*BLEU*	*Size*	*BLEU*
Small	38M	25.45	7.7M	17.35
Base	98M	27.11	9.0M	17.90
Big	284M	28.40	9.2M	17.89

Table 6.2: Statistics of NMT models and the corresponding phrase tables for different model capacities.

phrase pairs share the same source phrase as the "Shared" phrase table, of which 83.2% have a lower translation probability and has low quality. The results empirically confirm our hypothesis that *NMT models distill the bilingual knowledge by discarding those low-quality phrase pairs.*

6.4.4 Revisiting Recent Advances

In this section, we revisit recent advances that potentially affect the learning of bilingual knowledge. Specifically, we investigate three types of techniques: (1) *model capacity* that indicates how complicated patterns a model can express; (2) *data augmentation* that introduces additional knowledge with external data; and (3) *domain adaptation* that transfers knowledge across different domains.

Model	Shared		Non-Shared	
	Size	*BLEU*	*Size*	*BLEU*
SMALL	7.0M	17.53	0.7M	2.37
BASE	7.0M	17.49	2.0M	3.57
BIG	7.0M	17.29	2.2M	3.47

Table 6.3: Comparison among phrase tables that are extracted from models of different model capacities.

1) Model Capacity

We vary the layer dimensionality of Transformer, and obtain three model variants: SMALL (256), BASE (512), and BIG (1024). As listed in Table 6.2, increasing model capacity consistently improves translation performance. However, the extracted phrase table is only marginally increased.
We compare the phrase tables learned by different models, as shown in Table 6.3. The phrase table shared by all models takes the overwhelming majority, which adds most value to the translation performance ("Shared" BLEU). We conjecture that enlarging capacity improves NMT performance by better exploiting complex patterns beyond bilingual lexicons. It also confirms our intuition that bilingual lexicons can be a crucial early step in assessing the knowledge in NMT models.

2) Data Augmentation

In this experiment, we investigate two representative data augmentation approaches, i.e., back-translation [129] and forward-translation [165], which differ at exploiting target or source-side monolingual data, respectively. Specifically, we randomly sample a same-size (around 4.5M) English and German monolingual dataset from the WMT website, and construct the synthetic

Model	NMT		Phrase Table	
	#Para	*BLEU*	*Size*	*BLEU*
BASE	98M	27.11	9.0M	17.90
+ BT	98M	29.75	20.9M	19.26
+ FT	98M	28.43	28.0M	19.33

Table 6.4: NMT models and phrase tables for back-translation ("BT") and forward-translation ("FT").

Model	Shared		Non-Shared	
	Size	*BLEU*	*Size*	*BLEU*
BASE	8.3M	17.67	0.7M	1.78
+ BT	8.3M	18.61	12.6M	10.45
BASE	8.4M	17.83	0.5M	1.21
+ FT	8.4M	18.30	19.6M	11.25

Table 6.5: Comparison of phrase tables for BT and FT.

corpus with BASE models that are trained on the parallel data. Table 6.4 lists the results of NMT models and the extracted phrase tables. As shown, both data augmentation techniques significantly improve the performance of NMT models by exploiting a larger and better phrase table.[5] Besides, Table 6.5 shows the detailed comparison of the extracted phrase tables. Both augmentation methods *induce new knowledge and enhance existing knowledge over the baseline*, and the newly introduced knowledge contributes a lot to the translation performance improvement.

We further analyze the characteristics of the newly introduced phrase pairs in terms of different metrics, as illustrated in Figure 6.8. One interesting finding is that the newly introduced phrase pairs are notably longer than the original ones in the

[5]The different sizes of BT and FT phrase tables are due to the different monolingual datasets used for them, the averaged phrase length of which are 24.8 and 28.4, respectively.

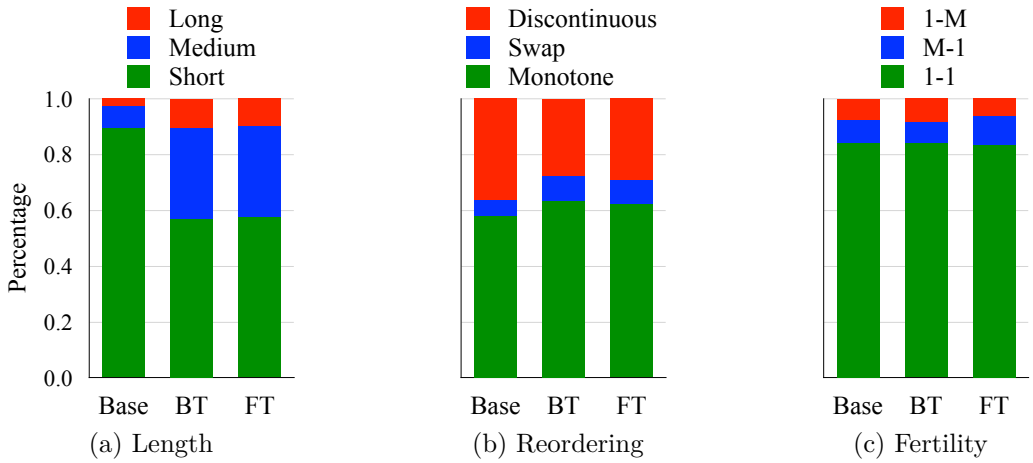


Figure 6.8: Distribution of metrics on phrases in base model and newly introduced by BT and FT.

base model. Besides, the new phrase pairs show less reordered patterns and more monotone patterns, which may explain the producing of longer phrases. The finding is consistent with previous studies, which show that the BT text is simpler than naturally occurring text [46].

3) Domain Adaptation

At last, we analyze the transferability of the bilingual knowledge of NMT models by directly applying it to another domain. To this end, we fine-tune the NMT model which was previously trained on the WMT14 En⇒De dataset (News, out-of-domain), on the IWSLT14 En⇒De dataset (Spoken language with 160K training examples, in-domain) for several epochs. We extract the phrase table using the in-domain training corpus, and the results are shown in Table 6.6. The fine-tuned NMT model benefits from a larger and better phrase table, by adapting the original model to the in-domain dataset. The analysis results in Table 6.7 further show that the fine-tuning technique improves

Fine	NMT		Phrase Table	
Tune	*# Para.*	*BLEU*	*Size*	*BLEU*
×	98M	15.78	168K	16.08
✓	98M	31.26	316K	18.50

Table 6.6: Statistics of NMT models and the corresponding phrase tables for the domain adaptation.

Fine	Shared		Non-Shared	
Tune	*Size*	*BLEU*	*Size*	*BLEU*
×	0.16M	15.95	0.01M	1.65
✓	0.16M	16.92	0.16M	6.95

Table 6.7: Comparison of phrase tables extracted from models with/without fine-tuning in domain adaptation.

performance with both more phrases ("Non-Shared") in the phrase table and better quality estimation of the original phrases ("Shared").

In addition, we re-extract the phrase table based on the out-of-domain dataset using the fine-tuned model. The phrase table achieves only a BLEU score of 4.77 with 2.6M phrase pairs, while the original phrase table without fine-tuning has 9.0M phrase pairs and its BLEU score is 17.90. To conclude, the fine-tuning approach increases new in-domain knowledge while forgetting certain amount of out-of-domain knowledge that are previously learned. The results provide an empirical validation of the *catastrophic forgetting* phenomenon in domain adaptation [81], which further demonstrate the reasonableness of our proposed approach.

6.5 Discussion

In this chapter, we propose to assess the bilingual knowledge learned by NMT models with statistic models – phrase table. The reported results provide a better understanding of NMT models and recent technological advances in learning the essential bilingual lexicons. The findings also indicate several potential applications which we leave for future exploration:

- *Error diagnosis* that debugs mistaken predictions by tracing associated phrase pairs [42];
- *Curriculum learning* that dynamically assigns more weights to instances associated with the unlearned knowledge [119];
- *Phrase memory* that stores unlearned phrases in NMT to query when generating translations [153, 164].

Although the phrase table successfully explains many model behaviors, it cannot explain certain techniques such as enlarging model capacity. The explored bilingual lexicon is only one of the critical knowledge bases in the translation process. In the future, we will investigate more advanced forms of bilingual knowledge [92, 56], as well as exploring other types of knowledge bases such as grammar with statistic models (e.g., reordering and language models). This chapter is the first step in what we hope will be a long and fruitful journey.

6.6 Summary

In this chapter, we propose to interpret the bilingual knowledge learned in the NMT models, which provides a global interpretation that help us better understand the model. Through exten-

sive experiments on different language pairs, model structures and random seeds, our study demonstrates the reasonableness and effectiveness of assessing the NMT knowledge with statistic models, which opens up a new angle to interpret NMT models. Equipped with the interpretable phrase table, we find that NMT models learn patterns from simple to complex and distill essential bilingual knowledge from the training examples. We also revisit several advances (e.g., back-translation) that potentially affect the learning of bilingual knowledge, and report some interesting findings. We believe this work opens up a new angle to interpret NMT with statistic models and provides empirical supports for recent advances in improving NMT models.

□ **End of chapter.**

Chapter 7

Conclusion and Future Work

7.1 Conclusion

As the software occupies our daily life and brings us convenience in various forms, its reliability is vital to both end-users and service providers. However, the increasing complexity and scale in both traditional software and intelligent software make them hard to understand, posing significant challenges for software reliability engineering. In this thesis, we study the software reliability engineering from the interpretability perspective, which is a crucial early step in realizing the software reliability. To be specific, in traditional software, we approach software reliability by interpreting the software logs for anomaly detection and problem identification. In intelligent software, we aim to interpret the model by estimating the input-output correspondence and assessing bilingual knowledge, respectively. The contributions are summarized as follows:

In Chapter 3, to fill the gap between the industry and academia on log-based anomaly detection, we provide the first systematic and comprehensive experience report. To achieve so, we evaluate six state-of-the-art anomaly detection methods on two

representative log datasets. Besides, we release an open-source toolkit of these anomaly detection methods for easy reuse and further study.

In Chapter 4, to tackle the challenges of the high imbalance of log distribution and the lack of labeled data, we propose a novel framework to identify impactful problems, i.e., Log3C. At its core is the cascading clustering algorithm, an efficient and effective clustering method on log sequences. Experiments on three real-world log data and real-world usage in online service systems confirm our method's effectiveness and efficiency.

In Chapter 5, we open the black box of intelligent software by the input-output attribution. Specifically, we employ the integrated gradients method to calculate the word importance. Extensive experiments on various configurations confirm that our method can outperform existing interpretation methods. Besides, we apply our approach to detect the under-translation error. Our linguistic analyses provide some interesting findings that can guide future model structure design.

In Chapter 6, to provide a global explanation of the intelligent software behaviors, we approach the model interpretability by assessing the bilingual knowledge. We propose to use the phrase table to represent bilingual knowledge. Massive experiments show that the phrase table is reasonable and consistent. We also obtain some interesting findings in model learning dynamics and model improvement methods based on the phrase table.

In summary, this thesis studies the reliability engineering of both traditional software and intelligent software from the interpretability perspective. Extensive experiments on widely-recognized datasets confirm the effectiveness and efficiency of our proposed methods.

7.2 Future Work

Software reliability engineering has been a long-standing research topic, which is recently advanced and enriched by artificial intelligence from two aspects: traditional software reliability with intelligent methods and intelligent software reliability. Although a number of novel techniques have been proposed from the interpretability perspective, there are plenty of exciting research directions that we leave for future work.

7.2.1 Advanced Log Analysis for Reliability Engineering

Due to the lack of a large amount of labeled data in traditional software reliability engineering, most existing researches focus on conventional machine learning techniques such as the decision tree and clustering. They did not take the full use of the power of more advanced learning methods, i.e., deep neural networks. Recently, the self-supervised learning [58, 113, 69, 36], which is also termed as representation learning, has demonstrated its ability in capturing patterns without labels. The idea behind is to learn the intrinsic properties or structures of the training data, which can alternatively serve the role of "label". For example, BERT [36] is a self-supervised learning method which has now become the state-of-the-art model for many NLP tasks. The low dependency on labeled data makes the self-supervised learning suitable for the log analysis, which would be a promising research direction in the next few years. Specifically, we can leverage the self-supervised learning techniques in log analysis by modeling the log sequences with RNN or Transformer structure. Intuitively, the model can learn the majority patterns of the log

sequences and leave the rest as anomalies.

7.2.2 Multi-source Intelligent Reliability Engineering

In this thesis, we approach the traditional software reliability by learning mainly from the interpretable log data. However, in the maintenance of industrial software systems, it is common to inspect many aspects of the software system, e.g., CPU utilization rate, memory consumption, etc. These monitoring metrics can interpret software behaviors, which could then help the automatic identification of software problems. We believe the use of multi-source interpretation data would be the mainstream of future research.

There are several challenges that need to be addressed. The first challenge is that the multi-source data might have too many types, which ones to consider in the modeling then becomes a problem. The second challenge to address is the heterogeneous [171] property for multi-source data, which makes the model design difficult.

7.2.3 Intelligent Software Robustness

Model robustness is at the core of ensuring intelligent software reliability. Recent studies [59, 6, 107] have shown that the deep learning models are vulnerable and brittle when feeding adversarial examples. How to make sure that deep learning models are robust against different perturbations is now a hot research topic [97, 121, 154].

We are interested in designing novel defense methods to protect the model from being attacked with the model interpretability. Since important input features contribute mostly to the model

prediction, these features should be explicitly protected with tailored strategies. On the contrary, other features play less critical roles in the model prediction, thereby they can be ignored naturally. Therefore, the interpretability could provide guidance in the model defense, which we believe is a promising direction.

□ **End of chapter.**

Chapter 8

Publications during Ph.D. Study

1. **Shilin He**, Jieming Zhu, Pinjia He, Michael R. Lyu. "Experience Report: System Log Analysis for Anomaly Detection". In Proceedings of the 27th International Symposium on Software Reliability Engineering (ISSRE), 2016.

2. Pinjia He, Jieming Zhu, **Shilin He**, Jian Li, Michael R. Lyu. "Towards Automated Log Parsing for Large-Scale Log Data Analysis". IEEE Transactions on Dependable and Secure Computing (TDSC), 2017.

3. **Shilin He**, Qingwei Lin, Jianguang Lou, Hongyu Zhang, Michael R. Lyu, Dongmei Zhang. "Identifying Impactful Service System Problems via Log Analysis". In Proceedings of the 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), 2018.

4. Pinjia He, Zhuangbin Chen, **Shilin He**, Michael R. Lyu. "Characterizing the Natural Language Descriptions in Software Logging Statements". In Proceedings of the 33rd

IEEE/ACM International Conference on Automated Software Engineering (ASE), 2018.

5. Jieming Zhu, **Shilin He**, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, Michael R. Lyu. "Tools and Benchmarks for Automated Log Parsing". In Proceedings of the 41st ACM/IEEE International Conference on Software Engineering (ICSE), 2019.

6. **Shilin He**, Zhaopeng Tu, Xing Wang, Longyue Wang, Michael R. Lyu, Shuming Shi. "Towards Understanding Neural Machine Translation with Word Importance". In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), 2019.

7. **Shilin He**, Xing Wang, Shuming Shi, Michael R. Lyu, Zhaopeng Tu. "Assessing the Bilingual Knowledge Learned by Neural Machine Translation Models". (In Submission).

8. **Shilin He**, Yongchang Hao, Xing Wang, Shuming Shi, Michael R. Lyu, Zhaopeng Tu. "Multi-Task Learning with Auxiliary Autoregressive Decoder for Non-Autoregressive Machine Translation". (In Submission).

Note: The papers [1, 3, 6, 7] are partially involved in this thesis.

□ **End of chapter.**

Bibliography

[1] Apache hadoop (http://hadoop.apache.org/).

[2] Apache spark (http://spark.apache.org/).

[3] Logentries: Log management & analysis software made easy (https://www.loggly.com/docs/anomaly-detection).

[4] Loggly: Cloud log management service (https://www.loggly.com/docs/anomaly-detection).

[5] O. A. Abbas. Comparisons between data clustering algorithms. *International Arab Journal of Information Technology (IAJIT)*, 5(3), 2008.

[6] N. Akhtar and A. Mian. Threat of adversarial attacks on deep learning in computer vision: A survey. *IEEE Access*, 6:14410–14430, 2018.

[7] J. Alonso, L. Belanche, and D. R. Avresky. Predicting software anomalies using machine learning techniques. In *NCA'11: Proc. of the 10th IEEE International Symposium on Network Computing and Applications*, pages 163–170. IEEE, 2011.

[8] D. Alvarez-Melis and T. Jaakkola. A causal framework for explaining the predictions of black-box sequence-to-sequence models. In *EMNLP*, 2017.

[9] T. W. Anderson, T. W. Anderson, T. W. Anderson, T. W. Anderson, and E.-U. Mathématicien. *An introduction to multivariate statistical analysis*, volume 2. Wiley New York, 1958.

[10] L. Arras, F. Horn, G. Montavon, K.-R. Müller, and W. Samek. Explaining predictions of non-linear classifiers in nlp. In *Proceedings of the 1st Workshop on Representation Learning for NLP*, 2016.

[11] D. R. Azevedo, A. M. Ambrósio, and M. Vieira. Applying data mining for detecting anomalies in satellites. In *EDCC'12: Proc. of the Ninth European Dependable Computing Conference*, pages 212–217. IEEE, 2012.

[12] A. Babenko, L. Mariani, and F. Pastore. Ava: automated interpretation of dynamically detected anomalies. In *ISSTA'09: Proc. of the eighteenth international symposium on Software testing and analysis*, pages 237–248. ACM, 2009.

[13] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140, 2015.

[14] A. Baevski and M. Auli. Adaptive input representations for neural language modeling. In *ICLR*, 2019.

[15] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. In *ICLR*, 2015.

[16] S. Banerjee, H. Srikanth, and B. Cukic. Log-based reliability analysis of software as a service (saas). In *ISSRE'10: Proc. of the 21st IEEE International Symposium on Software Reliability Engineering*, pages 239–248. IEEE, 2010.

[17] A. Bau, Y. Belinkov, H. Sajjad, N. Durrani, F. Dalvi, and J. Glass. Identifying and controlling important neurons in neural machine translation. In *ICLR*, 2019.

[18] Y. Belinkov, N. Durrani, F. Dalvi, H. Sajjad, and J. Glass. What do neural machine translation models learn about morphology? In *ACL*, 2017.

[19] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *ICML*, 2009.

[20] I. Beschastnikh, Y. Brun, M. D. Ernst, and A. Krishnamurthy. Inferring models of concurrent systems from logs of their behavior with csight. In *Proceedings of the 36th International Conference on Software Engineering*, pages 468–479. ACM, 2014.

[21] I. Beschastnikh, Y. Brun, S. Schneider, M. Sloan, and M. Ernst. Leveraging existing instrumentation to automatically infer invariant-constrained models. In *ESEC/FSE'11: Proc. of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, 2011.

[22] P. Bodik, M. Goldszmidt, A. Fox, D. B. Woodard, and H. Andersen. Fingerprinting the datacenter: automated classification of performance crises. In *EuroSys'10: Proc.*

of the 5th European conference on Computer systems, pages 111–124. ACM, 2010.

[23] A. Bovenzi, F. Brancati, S. Russo, and A. Bondavalli. An os-level framework for anomaly detection in complex software systems. *IEEE Transactions on Dependable and Secure Computing*, 12(3):366–372, 2015.

[24] P. F. Brown, J. Cocke, S. A. Della Pietra, V. J. Della Pietra, F. Jelinek, J. D. Lafferty, R. L. Mercer, and P. S. Roossin. A statistical approach to machine translation. *Computational linguistics*, 1990.

[25] P. F. Brown, V. J. D. Pietra, S. A. D. Pietra, and R. L. Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational linguistics*, 1993.

[26] P. F. Brown, V. J. D. Pietra, S. A. D. Pietra, and R. L. Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational linguistics*, 1993.

[27] M. Chen, A. X. Zheng, J. Lloyd, M. I. Jordan, and E. Brewer. Failure diagnosis using decision trees. In *ICAC'04: Proc. of the 1st International Conference on Autonomic Computing*, pages 36–43. IEEE, 2004.

[28] X. Chen, C. Lu, and K. Pattabiraman. Predicting job completion times using system logs in supercomputing clusters. In *in DSN-W'13: Proc. of the 43rd Annual IEEE/IFIP Conference on Dependable Systems and Networks Workshop (DSN-W)*, pages 1–8. IEEE, 2013.

[29] D. Chiang. A hierarchical phrase-based model for statistical machine translation. In *ACL*, 2005.

[30] N. Chomsky and D. W. Lightfoot. *Syntactic structures*. Walter de Gruyter, 2002.

[31] R. Christensen and F. Li. Adaptive log compression for massive log data. In *SIGMOD Conference*, pages 1283–1284, 2013.

[32] M. Cinque, D. Cotroneo, R. D. Crte, and A. Pecchia. What logs should you look at when an application fails? insights from an industrial case study. In *DSN'14: Proc. of the 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 690–695. IEEE, 2014.

[33] T. Cohn and M. Lapata. Sentence compression beyond word deletion. In *COLING*, 2008.

[34] A. Conneau, G. Kruszewski, G. Lample, L. Barrault, and M. Baroni. What you can cram into a single vector: Probing sentence embeddings for linguistic properties. *arXiv preprint arXiv:1805.01070*, 2018.

[35] Y. Dang, R. Wu, H. Zhang, D. Zhang, and P. Nobel. Rebucket: a method for clustering duplicate crash reports based on call stack similarity. In *Proceedings of the 34th International Conference on Software Engineering*, pages 1084–1093. IEEE Press, 2012.

[36] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019*

Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pages 4171–4186, 2019.

[37] K. Dhamdhere, M. Sundararajan, and Q. Yan. How important is a neuron? In *ICLR*, 2019.

[38] W. Dickinson, D. Leon, and A. Podgurski. Finding failures by cluster analysis of execution profiles. In *Proceedings of the 23rd international conference on Software engineering*, pages 339–348. IEEE Computer Society, 2001.

[39] N. DiGiuseppe and J. A. Jones. Software behavior and failure clustering: An empirical study of fault causality. In *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on*, pages 191–200. IEEE, 2012.

[40] R. Ding, Q. Fu, J. Lou, Q. Lin, D. Zhang, J. Shen, and T. Xie. Healing online service systems via mining historical issue repositories. In *ASE'12: Proc. of the 27th IEEE/ACM International Conference on Automated Software Engineering*, pages 318–321. IEEE, 2012.

[41] R. Ding, Q. Fu, J. G. Lou, Q. Lin, D. Zhang, and T. Xie. Mining historical issue repositories to heal large-scale online service systems. In *Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on*, pages 311–322. IEEE, 2014.

[42] Y. Ding, Y. Liu, H. Luan, and M. Sun. Visualizing and understanding neural machine translation. In *ACL*, 2017.

[43] T. Domhan. How much attention do you need? a granular analysis of neural machine translation architectures. In *ACL*, 2018.

[44] N. Durrani, B. Haddow, P. Koehn, and K. Heafield. Edinburgh's phrase-based machine translation systems for wmt-14. In *WMT*, 2014.

[45] C. Dyer, V. Chahuneau, and N. A. Smith. A simple, fast, and effective reparameterization of ibm model 2. In *NAACL*, 2013.

[46] S. Edunov, M. Ott, M. Ranzato, and M. Auli. On the evaluation of machine translation systems trained with back-translation. *arXiv preprint arXiv:1908.05204*, 2019.

[47] J. L. Elshoff and M. Marcotty. Improving computer program readability to aid modification. *Commun. ACM*, 1982.

[48] A. Fahad, N. Alshatri, Z. Tari, A. Alamri, I. Khalil, A. Y. Zomaya, S. Foufou, and A. Bouras. A survey of clustering algorithms for big data: Taxonomy and empirical analysis. *IEEE transactions on emerging topics in computing*, 2(3):267–279, 2014.

[49] M. Farshchi, J. Schneider, I. Weber, and J. Grundy. Experience report: Anomaly detection of cloud application operations using log and cloud metric correlation analysis. In *ISSRE'15: Proc. of the 26th International Symposium on Software Reliability Engineering*. IEEE, 2015.

[50] R. C. Fong and A. Vedaldi. Interpretable explanations of black boxes by meaningful perturbation. In *Proceedings of*

the IEEE International Conference on Computer Vision, pages 3429–3437, 2017.

[51] L. D. Fosdick and L. J. Osterweil. Data flow analysis in software reliability. *ACM Computing Surveys (CSUR)*, 8(3):305–330, 1976.

[52] G. Foster, P. Isabelle, and P. Plamondon. Target-text mediated interactive machine translation. *Machine Translation*, 12(1/2):175–194, 1997.

[53] Q. Fu, J. Lou, Y. Wang, and J. Li. Execution anomaly detection in distributed systems through unstructured log analysis. In *ICDM'09: Proc. of International Conference on Data Mining*, 2009.

[54] Q. Fu, J. Zhu, W. Hu, J. Lou, R. Ding, Q. Lin, D. Zhang, and T. Xie. Where do developers log? an empirical study on logging practices in industry. In *ICSE'14: Companion Proc. of the 36th International Conference on Software Engineering*, pages 24–33, 2014.

[55] X. Fu, R. Ren, J. Zhan, W. Zhou, Z. Jia, and G. Lu. Logmaster: mining event correlations in logs of large-scale cluster systems. In *SRDS'12: Proc. of the 31st IEEE Symposium on Reliable Distributed Systems*, pages 71–80. IEEE, 2012.

[56] M. Galley and C. D. Manning. Accurate non-hierarchical phrase-based translation. In *NAACL*, 2010.

[57] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin. Convolutional sequence to sequence learning. In *ICML*, 2017.

[58] S. Gidaris, P. Singh, and N. Komodakis. Unsupervised representation learning by predicting image rotations. In *International Conference on Learning Representations*, 2018.

[59] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[60] É. Grave, A. Joulin, M. Cissé, D. Grangier, and H. Jégou. Efficient softmax approximation for GPUs. In *ICML*, 2017.

[61] K. Gulordava, P. Bojanowski, E. Grave, T. Linzen, and M. Baroni. Colorless green recurrent networks dream hierarchically. In *NAACL*, 2018.

[62] J. Guo, X. Tan, D. He, T. Qin, L. Xu, and T.-Y. Liu. Non-autoregressive neural machine translation with enhanced decoder input. In *AAAI*, 2019.

[63] R. H. Hahnloser, R. Sarpeshkar, M. A. Mahowald, R. J. Douglas, and H. S. Seung. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405(6789):947–951, 2000.

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

[65] J. Han, M. Kamber, and J. Pei. *Data mining: concepts and techniques*. Elsevier, 2011.

[66] H. Hassan, A. Aue, C. Chen, V. Chowdhary, J. Clark, C. Federmann, X. Huang, M. Junczys-Dowmunt, W. Lewis, M. Li, et al. Achieving human parity on automatic chinese to english news translation. In *arXiv:1803.05567*, 2018.

[67] F. Hayes-Roth. Rule-based systems. *Communications of the ACM*, 28(9):921–932, 1985.

[68] J. He, J. Gu, J. Shen, and M. Ranzato. Revisiting self-training for neural sequence generation. In *ICLR*, 2020.

[69] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9729–9738, 2020.

[70] P. He, J. Zhu, S. He, J. Li, and R. Lyu. An evaluation study on log parsing and its use in log mining. In *DSN'16: Proc. of the 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2016.

[71] 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.

[72] S. He, Z. Tu, X. Wang, L. Wang, M. Lyu, and S. Shi. Towards understanding neural machine translation with word importance. In *EMNLP*, 2019.

[73] K. Heafield. KenLM: faster and smaller language model queries. In *EMNLP 2011 Sixth Workshop on Statistical Machine Translation*, 2011.

[74] J. D. Herbsleb and A. Mockus. An empirical study of speed and communication in globally distributed software development. *IEEE Trans. Softw. Eng.*, 29(6):481–494, June 2003.

[75] F. Hill, K. Cho, S. Jean, and Y. Bengio. The representational geometry of word meanings acquired by neural machine translation models. *Machine Translation*, 31(1-2):3–18, 2017.

[76] C. Hokamp and Q. Liu. Lexically constrained decoding for sequence generation using grid beam search. In *ACL*, 2017.

[77] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM computing surveys (CSUR)*, 31(3):264–323, 1999.

[78] S. Jain and B. C. Wallace. Attention is not explanation. In *NAACL*, 2019.

[79] Z. M. Jiang, A. E. Hassan, P. Flora, and G. Hamann. Abstracting execution logs to execution events for enterprise applications (short paper). In *Quality Software, 2008. QSIC'08. The Eighth International Conference on*, pages 181–186. IEEE, 2008.

[80] H. Johnson, J. Martin, G. Foster, and R. Kuhn. Improving translation quality by discarding most of the phrasetable. In *EMNLP*, 2007.

[81] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, et al. Overcoming catas-

trophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 2017.

[82] T. Kocmi and O. Bojar. Curriculum learning and minibatch bucketing in neural machine translation. In *RANLP*, 2017.

[83] P. Koehn. *Statistical machine translation*. Cambridge University Press, 2009.

[84] P. Koehn, H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens, et al. Moses: Open source toolkit for statistical machine translation. In *ACL*, 2007.

[85] P. Koehn, F. J. Och, and D. Marcu. Statistical phrase-based translation. In *NAACL*, 2003.

[86] P. W. Koh and P. Liang. Understanding black-box predictions via influence functions. *arXiv preprint arXiv:1703.04730*, 2017.

[87] G. Lample, M. Ott, A. Conneau, L. Denoyer, and M. Ranzato. Phrase-based & neural unsupervised machine translation. In *EMNLP*, 2018.

[88] S. Läubli, R. Sennrich, and M. Volk. Has Machine Translation Achieved Human Parity? A Case for Document-level Evaluation. In *EMNLP*, Aug. 2018.

[89] Y. Liang, Y. Zhang, H. Xiong, and R. Sahoo. Failure prediction in ibm bluegene/l event logs. In *ICDM'07: Proc. of the 7th International Conference on Data Mining*, 2007.

[90] Q. Lin, H. Zhang, J. Lou, Y. Zhang, and X. Chen. Log clustering based problem identification for online service systems. In *ICSE'16: Proc. of the 38th International Conference on Software Engineering*, 2016.

[91] J. Liu, J. Zhu, S. He, P. He, Z. Zheng, and M. R. Lyu. Logzip: extracting hidden structures via iterative clustering for log compression. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 863–873. IEEE, 2019.

[92] Y. Liu, Q. Liu, and S. Lin. Tree-to-string alignment template for statistical machine translation. In *ACL*, 2006.

[93] J. Lou, Q. Fu, S. Yang, Y. Xu, and J. Li. Mining invariants from console logs for system problem detection. In *ATC'10: Proc. of the USENIX Annual Technical Conference*, 2010.

[94] X. Lu. Automatic analysis of syntactic complexity in second language writing. *International journal of corpus linguistics*, 2010.

[95] S. M. Lundberg and S.-I. Lee. A unified approach to interpreting model predictions. In *Advances in neural information processing systems*, pages 4765–4774, 2017.

[96] M.-T. Luong and C. D. Manning. Stanford neural machine translation systems for spoken language domains. In *IWSLT*, 2015.

[97] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards deep learning models resistant to

adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[98] A. Makanju, A. Zincir-Heywood, and E. Milios. Clustering event logs using iterative partitioning. In *KDD'09: Proc. of International Conference on Knowledge Discovery and Data Mining*, 2009.

[99] C. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[100] L. Mariani and F. Pastore. Automated identification of failure causes in system logs. In *Software Reliability Engineering, 2008. ISSRE 2008. 19th International Symposium on*, pages 117–126. IEEE, 2008.

[101] H. Mi, H. Wang, Y. Zhou, R. Lyu, and H. Cai. Toward fine-grained, unsupervised, scalable performance diagnosis for production cloud computing systems. *IEEE Transactions on Parallel and Distributed Systems*, 24:1245–1255, 2013.

[102] A. Milenkoski, M. Vieira, S. Kounev, A. Avritzer, and B. Payne. Evaluating computer intrusion detection systems: A survey of common practices. *ACM Computing Surveys (CSUR)*, 48(1):12, 2015.

[103] G. Montavon, S. Lapuschkin, A. Binder, W. Samek, and K.-R. Müller. Explaining nonlinear classification decisions with deep taylor decomposition. *Pattern Recognition*, 65:211–222, 2017.

[104] M. Morishita, J. Suzuki, and M. Nagata. Ntt neural machine translation systems at wat 2017. In *WAT*, 2017.

[105] P. K. Mudrakarta, A. Taly, M. Sundararajan, and K. Dhamdhere. Did the model understand the question? In *ACL*, 2018.

[106] G. Neubig, M. Morishita, and S. Nakamura. Neural reranking improves subjective quality of machine translation: Naist at wat2015. In *WAT*, 2015.

[107] A. Nguyen, J. Yosinski, and J. Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 427–436, 2015.

[108] J. Niehues and E. Cho. Exploiting linguistic resources for neural machine translation using multi-task learning. *arXiv preprint arXiv:1708.00993*, 2017.

[109] F. J. Och. Minimum error rate training in statistical machine translation. In *ACL*, 2003.

[110] F. J. Och and H. Ney. The alignment template approach to statistical machine translation. *CL*, 30(4):417–449, 2004.

[111] A. Oliner, A. Ganapathi, and W. Xu. Advances and challenges in log analysis. *Communications of the ACM*, 55(2):55–61, 2012.

[112] A. Oliner and J. Stearley. What supercomputers say: A study of five system logs. In *DSN'07:Proc. of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2007.

[113] A. v. d. Oord, Y. Li, and O. Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[114] M. Ott, S. Edunov, A. Baevski, A. Fan, S. Gross, N. Ng, D. Grangier, and M. Auli. Fairseq: A fast, extensible toolkit for sequence modeling. *NAACL*, 2019.

[115] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. Bleu: a method for automatic evaluation of machine translation. In *ACL*, 2002.

[116] A. Pecchia, M. Cinque, G. Carrozza, and D. Cotroneo. Industry practices and event logging: assessment of a critical software development process. In *ICSE'15: Proc. of the 37th International Conference on Software Engineering*, pages 169–178, 2015.

[117] A. Pecchia, D. Cotroneo, Z. Kalbarczyk, and R. Iyer. Improving log-based field failure data analysis of multi-node computing systems. In *DSN'11: Proc. of the 41st IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 97–108. IEEE, 2011.

[118] F. Pedregosa, G. Varoquaux, A. Gramfort, et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[119] E. A. Platanios, O. Stretcu, G. Neubig, B. Poczos, and T. Mitchell. Competence-based curriculum learning for neural machine translation. In *NAACL*, 2019.

[120] M. Post and D. Vilar. Fast lexically constrained decoding

with dynamic beam allocation for neural machine translation. In *NAACL*, 2018.

[121] A. Raghunathan, J. Steinhardt, and P. Liang. Certified defenses against adversarial examples. *arXiv preprint arXiv:1801.09344*, 2018.

[122] N. Rahaman, A. Baratin, D. Arpit, F. Draxler, M. Lin, F. Hamprecht, Y. Bengio, and A. Courville. On the spectral bias of neural networks. In *ICML*, 2019.

[123] M. T. Ribeiro, S. Singh, and C. Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *KDD*. ACM, 2016.

[124] S. Rifkin and L. Deimel. Applying program comprehension techniques to improve software inspections. 1994.

[125] G. Salton and C. Buckley. Term weighting approaches in automatic text retrival. Technical report, Cornell, 1987.

[126] S. Sato. Example-based machine translation. 1992.

[127] R. Schwarzenberg, D. Harbecke, V. Macketanz, E. Avramidis, and S. Möller. Train, sort, explain: Learning to diagnose translation models. In *NAACL*, 2019.

[128] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *ICCV*, 2017.

[129] R. Sennrich, B. Haddow, and A. Birch. Neural machine translation of rare words with subword units. In *ACL*, 2016.

[130] W. Shang, Z. M. Jiang, H. Hemmati, B. Adams, A. E. Hassan, and P. Martin. Assisting developers of big data analytics applications when deploying on hadoop clouds. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 402–411. IEEE Press, 2013.

[131] X. Shi, I. Padhi, and K. Knight. Does string-based neural mt learn source syntax? In *EMNLP*, 2016.

[132] E. Shihab, A. Mockus, Y. Kamei, B. Adams, and A. E. Hassan. High-impact defects: a study of breakage and surprise defects. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, pages 300–310. ACM, 2011.

[133] A. Shrikumar, P. Greenside, and A. Kundaje. Learning important features through propagating activation differences. In *ICML*, 2017.

[134] A. Shrikumar, P. Greenside, A. Shcherbina, and A. Kundaje. Not just a black box: Learning important features through propagating activation differences. *arXiv preprint arXiv:1605.01713*, 2016.

[135] M. R. Siegel and J. I. Ferrell. Method and system for determining software reliability, Aug. 20 1996. US Patent 5,548,718.

[136] S. E. Sim. *Supporting multiple program comprehension strategies during software maintenance.* University of Toronto, Department of Computer Science, 1998.

[137] Stanford. Evaluation of Clustering, NMI. https://nlp.stanford.edu/IR-book/html/htmledition/evaluation-of-clustering-1.html, 2008. [Online; accessed July-2018].

[138] E. Strubell, P. Verga, D. Andor, D. Weiss, and A. McCallum. Linguistically-Informed Self-Attention for Semantic Role Labeling. In *EMNLP*, 2018.

[139] M. Sundararajan and A. Najmi. The many shapley values for model explanation. *arXiv preprint arXiv:1908.08474*, 2019.

[140] M. Sundararajan, A. Taly, and Q. Yan. Axiomatic attribution for deep networks. In *ICML*, 2017.

[141] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *NIPS*, 2014.

[142] L. Tang, T. Li, and C. Perng. LogSig: generating system events from raw textual logs. In *CIKM'11: Proc. of ACM International Conference on Information and Knowledge Management*, pages 785–794, 2011.

[143] C. Tillmann. A unigram orientation model for statistical machine translation. In *HLT-NAACL*, 2004.

[144] M. Toneva, A. Sordoni, R. T. d. Combes, A. Trischler, Y. Bengio, and G. J. Gordon. An empirical study of example forgetting during deep neural network learning. 2019.

[145] Z. Tu, Z. Lu, Y. Liu, X. Liu, and H. Li. Modeling coverage for neural machine translation. In *ACL*, 2016.

[146] R. Vaarandi. A data clustering algorithm for mining patterns from event logs. In *IPOM'03: Proc. of the 3rd Workshop on IP Operations and Management*, 2003.

[147] R. Vaarandi and M. Pihelgas. Logcluster-a data clustering and pattern mining algorithm for event logs. In *2015 11th International conference on network and service management (CNSM)*, pages 1–7. IEEE, 2015.

[148] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *NeurIPS*, 2017.

[149] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is All You Need. In *NIPS*, 2017.

[150] R. Venkatakrishnan and M. A. Vouk. Diversity-based detection of security anomalies. In *HotSoS'14: Proc. of the 2014 Symposium and Bootcamp on the Science of Security*, page 29. ACM, 2014.

[151] E. Voita, R. Sennrich, and I. Titov. The bottom-up evolution of representations in the transformer: A study with machine translation and language modeling objectives. In *EMNLP*, 2019.

[152] E. Voita, D. Talbot, F. Moiseev, R. Sennrich, and I. Titov. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *ACL*, 2019.

[153] X. Wang, Z. Lu, Z. Tu, H. Li, D. Xiong, and M. Zhang. Neural machine translation advised by statistical machine translation. In *AAAI*, 2017.

[154] T.-W. Weng, H. Zhang, P.-Y. Chen, J. Yi, D. Su, Y. Gao, C.-J. Hsieh, and L. Daniel. Evaluating the robustness of neural networks: An extreme value theory approach. *arXiv preprint arXiv:1801.10578*, 2018.

[155] S. Wiegreffe and Y. Pinter. Attention is not not explanation. In *EMNLP*, 2019.

[156] Wikipedia. Complete linkage clustering. https://en.wikipedia.org/wiki/Complete-linkage_clustering, 2018. [Online; accessed July-2018].

[157] Wikipedia. Multivariate normal distribution. https://en.wikipedia.org/wiki/Multivariate_normal_distribution, 2018. [Online; accessed July-2018].

[158] Wikipedia. Sigmoid Function. https://en.wikipedia.org/wiki/Sigmoid_function, 2018. [Online; accessed July-2018].

[159] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordon. Detecting large-scale system problems by mining console logs. In *SOSP'09: Proc. of the ACM Symposium on Operating Systems Principles*, 2009.

[160] L. Xuan, C. Zhigang, and Y. Fan. Exploring of clustering algorithm on class-imbalanced data. In *2013 8th International Conference on Computer Science Education*, pages 89–93, April 2013.

[161] Y. Yin, J. Su, H. Wen, J. Zeng, Y. Liu, and Y. Chen. Pos tag-enhanced coarse-to-fine attention for neural machine translation. *ACM Transactions on Asian and Low-Resource Language Information Processing (TALLIP)*, 18(4):1–14, 2019.

[162] C. Yuan, N. Lao, J.-R. Wen, J. Li, Z. Zhang, Y.-M. Wang, and W.-Y. Ma. Automated known problem diagnosis with event traces. In *ACM SIGOPS Operating Systems Review*, volume 40, pages 375–388. ACM, 2006.

[163] D. Yuan, S. Park, P. Huang, Y. Liu, M. M.-J. Lee, X. Tang, Y. Zhou, and S. Savage. Be conservative: Enhancing failure diagnosis with proactive logging. In *OSDI*, volume 12, pages 293–306, 2012.

[164] J. Zhang, Y. Liu, H. Luan, J. Xu, and M. Sun. Prior knowledge integration for neural machine translation using posterior regularization. In *ACL*, 2017.

[165] J. Zhang and C. Zong. Exploiting source-side monolingual data in neural machine translation. In *EMNLP*, 2016.

[166] W. E. Zhang, Q. Z. Sheng, and A. A. F. Alhazmi. Generating textual adversarial examples for deep learning models: A survey. In *arXiv preprint arXiv:1901.06796*, 2019.

[167] Y. Zhao, Y. Wang, J. Zhang, and C. Zong. Phrase table as recommendation memory for neural machine translation. In *IJCAI*, 2018.

[168] J. Zhu, P. He, Q. Fu, H. Zhang, M. R. Lyu, and D. Zhang. Learning to log: Helping developers make in-

formed logging decisions. In *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on*, volume 1, pages 415–425. IEEE, 2015.

[169] 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.

[170] L. M. Zintgraf, T. S. Cohen, T. Adel, and M. Welling. Visualizing deep neural network decisions: Prediction difference analysis. In *ICLR*, 2017.

[171] R. Zuech, T. M. Khoshgoftaar, and R. Wald. Intrusion detection and big heterogeneous data: a survey. *Journal of Big Data*, 2(1):3, 2015.