



Run Run Shaw Library

香港城市大學
City University of Hong Kong

Copyright Warning

Use of this thesis/dissertation/project is for the purpose of private study or scholarly research only. ***Users must comply with the Copyright Ordinance.***

Anyone who consults this thesis/dissertation/project is understood to recognise that its copyright rests with its author and that no part of it may be reproduced without the author's prior written consent.

CITY UNIVERSITY OF HONG KONG
香港城市大學

Deep Representation Learning in Bug Report
and Log Anomaly Detection
深度表示學習在缺陷報告
和日誌異常檢測中的應用

Submitted to
Department of Computer Science
電腦科學系
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy
哲學博士學位

by

MA Xiaoxue
馬笑雪

AUGUST 2024
二零二四年八月

Declaration

I declare that the thesis entitled “Deep Representation Learning in Bug Report and Log Anomaly Detection” represents my own work, except where due acknowledgments are made, and that it has not been previously included in a thesis, dissertation, or report submitted to this university or any other institution for a degree, diploma, or other qualifications.

MA Xiaoxue
AUGUST 2024

Abstract

The integration of deep learning techniques into bug report and log anomaly detection represents a foundational aspect of software and system maintenance. Typically, these analyses encompass two primary stages: text representation and representation learning. Text representation involves converting raw text data into machine-readable formats, such as embeddings in the feature space, while representation learning utilizes a variety of techniques, including machine learning and deep learning, to effectively capture these representation features. Furthermore, clustering and resampling techniques are commonly used to enhance representation learning by facilitating the identification of meaningful data patterns and addressing issues like class imbalance.

In this thesis, we embark on a comprehensive exploration of four pivotal software engineering tasks, each tackling crucial aspects of bug report analysis and log anomaly detection: (1) security bug report detection, (2) bug report title generation, (3) log anomaly detection, and (4) data resampling in log anomaly detection. We propose three novel deep learning approaches (CASMS, AttSum, AdaLog) and conduct an empirical study to address ongoing research challenges.

Specifically, bug reports (BRs) serve as indispensable records in software maintenance, documenting issues reported by users for resolution by software developers. Given the different severity of bug reports, prioritizing the detection of Security Bug Reports (SBRs) is critical. Identifying and resolving bugs exposed in SBRs can help mitigate potential damage to software products before disclosure. Moreover, the bug report title is a compulsory component of a BR, making the automatic generation of high-quality bug report titles a focal point for researchers. This task aims to effectively extract the core content of a BR, thereby improving its readability and usability. Logs play a pivotal role in maintaining software-intensive systems, capturing vital information essential for troubleshooting and performance monitoring. Consequently, the analysis of logs for anomaly detection emerges as a

significant area of interest. Despite numerous studies dedicated to these tasks in recent years, persistent and enduring research challenges continue to remain unresolved.

In the realm of security bug report detection, the inappropriate disclosure of SBRs poses significant risks to software systems. However, prevailing techniques face challenges arising from class imbalance, inadequate training data, and limited performance robustness. To address these issues, we introduce the CASMS approach. CASMS leverages weighted word embeddings derived from $tf-idf$ and $word2vec$ techniques to transform bug reports. Subsequently, it utilizes the Elbow method and k -means clustering algorithm to automatically select diverse non-security bug reports (NSBRs), with the aim of addressing the class imbalance. Finally, we employ an Attention CNN-BLSTM model to process both SBRs and selected NSBRs, efficiently extracting contextual and sequential information for representation learning.

Towards bug report title generation, the creation of concise and informative titles plays a crucial role in facilitating efficient bug resolution processes. However, the challenge lies in the ability of bug reporters to generate high-quality titles. To mitigate this, we propose AttSum, an innovative deep attention-based summarization model. AttSum employs an encoder-decoder framework, utilizing the robustly optimized bidirectional-encoder-representations-from-transformers (RoBERTa) approach for deep text representation of bug reports. Moreover, it incorporates a stacked transformer decoder to generate bug report title tokens. Both the RoBERTa encoder and the stacked transformer decoder utilize multi-head attention operations, enabling them to memorize previous information and capture long-range dependencies between input and output sequences, regardless of distance. Consequently, AttSum excels in extracting and summarizing the global semantic information of bug report bodies, enhancing its overall performance. Additionally, we employ a copy mechanism to address the rare-term problem effectively.

In the domain of log anomaly detection (LAD), a variety of machine learning and deep learning approaches have been proposed, categorized into supervised, semi-supervised, and unsupervised methods. While semi-supervised techniques show promise by requiring only a fraction of labeled data and demonstrating relative stability, current approaches often suffer from manual parameter tuning and high false positives. To address these challenges, we introduce AdaLog, an integrated semi-supervised approach. Specifically, AdaLog utilizes a pre-trained model for word embedding, employs a self-adaptive clustering method to accurately calculate label probabilities for unlabeled data across twelve designed scenarios, and integrates a transformer-based

model for prediction. Furthermore, to mitigate class imbalance, AdaLog incorporates undersampling to enhance model performance.

In the realm of LAD, the persistent issue of class imbalance in publicly available data hinders the effectiveness of deep learning-based models. Nonetheless, whether data resampling can address this imbalance remains uncertain. To bridge this gap, we conduct a comprehensive analysis of various data resampling techniques and their impact on existing deep learning-based log anomaly detection (DLLAD) approaches. Through empirical evaluations on multiple benchmarks, we elucidate the effectiveness of resampling techniques in alleviating class imbalance and improving the performance of DLLAD approaches. Our study offers a valuable roadmap for researchers aiming to tackle the challenge of data imbalance in DLLAD. By implementing the recommended strategies for data resampling, researchers can significantly improve the performance and effectiveness of DLLAD approaches.

In summary, this thesis introduces novel approaches and empirical studies geared toward enhancing bug report analysis and log anomaly detection in software and system maintenance. Through our proposed approaches and empirical insights, we aim to make meaningful contributions to the continual advancement of robust and effective techniques within these domains.

Keywords: Text Representation Learning, Security Bug Report Detection, Title Generation, Log Anomaly Detection, Data Resampling.

CITY UNIVERSITY OF HONG KONG
Qualifying Panel and Examination Panel

Surname: MA
First Name: Xiaoxue
Degree: Doctor of Philosophy
College/Department: Department of Computer Science

The Qualifying Panel of the above student is composed of:

Supervisor(s)

Prof. KEUNG Wai Jacky Department of Computer Science
City University of Hong Kong

Qualifying Panel Member(s)

Prof. WANG Shiqi Department of Computer Science
City University of Hong Kong

Prof. SONG Linqi Department of Computer Science
City University of Hong Kong

This thesis has been examined and approved by the following examiners:

Prof. KEUNG Jacky Department of Computer Science
City University of Hong Kong

Prof. LEUNG Wing Ho Howard Department of Computer Science
City University of Hong Kong

Prof. WONG Hau San Department of Computer Science
City University of Hong Kong

Dr. LAI Richard Department of Computer Science and
Information Technology
La Trobe University

Acknowledgements

First and foremost, I want to express my heartfelt gratitude to my supervisor, Prof. Jacky Keung. Throughout my entire Ph.D. journey, he has been an unwavering source of invaluable guidance and support, lifting me up during moments of frustration and doubt. Even if outcomes didn't match our expectations sometimes, he consistently affirmed my accomplishments and expressed pride in my efforts. His words of encouragement provided me with immense comfort and served as a profound source of inspiration. Additionally, I extend my big thanks to Prof. Federica Sarro and Prof. Earl T. Barr for their inspiring mentorship during my visit time at University College London. Each of them has a unique approach to guiding students, with Federica's warmth akin to a nurturing sun and Earl's guidance reminiscent of a steadfast pine tree. Federica's encouragement and insightful suggestions have been instrumental in my growth, while Earl's exacting standards and rigorous approach to science have pushed me to strive for excellence. Despite their towering achievements, these Professors have graciously extended their wisdom and support to me, a student still finding my footing. Their generosity and dedication have been invaluable assets throughout my four-year journey, and I am deeply grateful for the priceless lessons I have learned under their guidance.

I also want to express my gratitude to my Ph.D. qualifying panel members, Prof. Shiqi Wang and Prof. Linqi Song, for their dedicated time and meticulous evaluation. Their thoughtful comments and insightful questions have played a pivotal role in enriching the depth and scope of my research. Furthermore, I am very grateful to my dissertation examiners, Prof. Hau San Wong, Prof. Howard Leung, and Prof. Richard Lai, for their invaluable feedback and suggestions. Their constructive insights have greatly inspired me and contributed to improving the quality of my thesis.

I am deeply thankful to my family – especially my dear parents, my uncles, and my beloved puppies, Kaikai and Potato. Throughout the past four years, their unwavering support has been my rock during challenging times. Whenever I am sad, or my confidence is shaken, they are there to accompany me, comfort me, and remind me

that the most important thing is not the results; as long as the journey is rewarding, it is worth it. Their continued faith in me means a lot to me.

Finally, I am so grateful for my besties Xiaodi Lu, Yihe Liu, and MaoGroup, my adorable lab mates Yishu Li, Jingyu Zhang, Yihan Liao, and Zhen Yang, and my collaborators Dr. Xiao Yu, Dr. Yan Xiao, and Dr. Pinjia He. They comforted me in my lowest moments and celebrated my successes with me. The Ph.D. journey has been both long and short, passionate yet monotonous. Without their company, I would not have been able to reach this point. I cherish our friendship and look to the future together.

*In the journey's stride, we forge ahead,
Yet glancing back, our bounty is spread.*

List of Publications

Published and Under Reviewed Journal Papers

1. **Xiaoxue Ma**, Huiqi Zou, Jacky Keung, Federica Sarro, Xiao Yu, Yishu Li, Zhen Yang, Jingyu Zhang. “On the Influence of Data Resampling for Deep Learning-Based Anomaly Detection: Insights and Recommendations.” *IEEE Transactions on Software Engineering (TSE)*, 2024. [Major Revision]
2. **Xiaoxue Ma**, Jacky Keung, Pinjia He, Xiao Yu, Yan Xiao, and Yishu Li. “A Semi-supervised Approach for Industrial Anomaly Detection via Self-adaptive Clustering.” *IEEE Transactions on Industrial Informatics (TII)*, 2023, doi: 10.1109/TII.2023.3280246.
3. **Xiaoxue Ma**, Jacky Keung, Xiao Yu, Huiqi Zou, Jingyu Zhang, and Yishu Li. “ATTSUM: A Deep Attention-Based Summarization Model for Bug Report Title Generation.” *IEEE Transactions on Reliability (TR)*, 2022, doi:10.1109/TR.2023.3236404.
4. **Xiaoxue Ma**, Jacky Keung, Zhen Yang, Xiao Yu, Yishu Li, and Hao Zhang. “CASMS: Combining clustering with attention semantic model for identifying security bug reports”. *Information and Software Technology (IST)*, 2022, 147:106906.
5. Jingyu Zhang, Jacky Keung, Yan Xiao, Yihan Liao, Yishu Li, **Xiaoxue Ma**. “UniAda: Universal Adaptive Multi-objective Adversarial Attack for End-to-End Autonomous Driving Systems.” *IEEE Transactions on Reliability (TR)*, 2024.
6. Yishu Li, Jacky Keung, Zhen Yang, **Xiaoxue Ma**, Jingyu Zhang, and Shuo Liu. “SimAC: Simulating Agile Collaboration to Generate Acceptance Criteria in User Story Elaboration.” *Automated Software Engineering Journal (ASE)*, 2024.

Published and Under Reviewed Conference Papers

1. Yanzhong He, Guancheng Lin, **Xiaoxue Ma***, Jacky Wai Keung, and Fuyang Li. “Exploring the impact of loss function on vulnerability detection”. *The 15th Asia-Pacific Symposium on Internetware, IEEE (Internetware 2024)*.
2. Zhen Yang, Fang Liu, Zhongxing Yu, Jia Li, Shuo Liu, Yifan Hong, **Xiaoxue Ma**, Jacky Keung, Zhi Jin, Ge Li. “Exploring and Unleashing the Power of Large Language Models in Automated Code Translation”. *In 2024 ACM International Conference on the Foundations of Software Engineering (FSE 2024)*.
3. Jingyu Zhang, Jacky Keung, **Xiaoxue Ma**, Xiangyu Li, Yan Xiao, Yishu Li, and Wing Kwong Chan. “Enhancing Valid Test Input Generation with Distribution Awareness for Deep Neural Networks”. *The 48th IEEE International Conference on Computers, Software, and Applications, IEEE (COMPSAC 2024)*.
4. Yishu Li, Jacky Keung, **Xiaoxue Ma**, Jingyu Zhang. “LLM-based Class Diagram Derivation from User Stories with Chain-of-Thought Prompting.” *The 48th IEEE International Conference on Computers, Software, and Applications, IEEE (COMPSAC 2024)*.
5. Yishu Li, Jacky Keung, Kwabena Ebo Bennin, **Xiaoxue Ma**, Jingyu Zhang, and Zhen Yang. “Agile Requirements Engineering in a Distributed Environment: Experiences from the Software Industry during Unprecedented Global Challenges.” *The 48th IEEE International Conference on Computers, Software, and Applications, IEEE (COMPSAC 2024)*.
6. Zexian Zhang, Shuang Yin, Wenliu Wei, **Xiaoxue Ma**, Jacky Wai Keung, Fuyang Li, and Wenhua Hu. “Practitioners’ expectations on code smell detection”. *The 48th IEEE International Conference on Computers, Software, and Applications, IEEE (COMPSAC 2024)*.
7. Yishu Li, Jacky Keung, **Xiaoxue Ma**. “Integrating Generative AI in Software Engineering Education: Practical Strategies” *The 10th International Symposium on Educational Technology, IEEE (ISET 2024)*.
8. Yishu Li, Jacky Keung, **Xiaoxue Ma**, Jingyu Zhang, Zhen Yang, and Shuo, Liu. “Learning Gaps in Project-based Requirements Engineering Education - A Case

- Study of Student Projects.” *The 9th International Symposium on Educational Technology*, pp.239-243, *IEEE (ISET 2023)*.
9. Yishu Li, Jacky Keung, Kwabena Ebo Bennin, **Xiaoxue Ma**, Yangyang Huang, and Jingyu Zhang. “Towards Requirements Engineering Activities for Machine Learning-enabled FinTech Applications” *The 30th Asia-Pacific Software Engineering Conference, IEEE (APSEC 2023)*.
 10. Zhen Yang, Jacky Keung, Xiao Yu, Xiaodong Gu, Zhengyuan Wei, **Xiaoxue Ma**, and Miao Zhang. “A multi-modal transformer-based code summarization approach for smart contracts”. *The 2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC)*, pp. 1-12. *IEEE*, 2021.

Table of contents

Declaration	ii
Abstract	iii
Qualifying Panel and Examination Panel	vi
Acknowledgements	vii
List of Publications	ix
List of figures	xvii
List of tables	xix
1 Introduction	1
1.1 Security Bug Report Detection	3
1.2 Bug Report Title Generation	5
1.3 Log Anomaly Detection	7
1.4 Data Resampling in Log Anomaly Detection	8
1.5 Contributions of Thesis	10
1.6 Organization of the Thesis	11
2 Literature Review	13
2.1 Security Bug Report Detection	13
2.2 Text Summarization	15
2.2.1 Extractive Methods	16
2.2.2 Abstractive Methods	16
2.3 Log Anomaly Detection	17
2.3.1 Supervised Approaches	17
2.3.2 Semi-Supervised and Unsupervised Approaches	19

2.4	Data Resampling	20
2.4.1	Data Resampling in Software Engineering	20
2.4.2	Data Resampling for Anomaly Detection	20
2.5	Summary	21
3	CASMS: Combining Clustering with Attention Semantic Model for Identifying Security Bug Reports	22
3.1	Background	23
3.1.1	Text Representation	23
3.1.2	Clustering	24
3.1.3	Models for Representation Learning	25
3.2	Methodology	27
3.2.1	Framework Overview	27
3.2.2	Data Preparation	27
3.2.3	Data Filtering	29
3.2.4	Model Training and Prediction	31
3.3	Experimental Setup	34
3.3.1	Research Questions	34
3.3.2	Datasets	34
3.3.3	Implementation	35
3.3.4	Evaluation Metrics	37
3.4	Results	39
3.4.1	RQ1. How effective is CASMS in classifying bug reports? . .	39
3.4.2	RQ2. How does CASMS find the best results effectively and how robust is it based on the number of clusters and the total number of selected NSBRs?	41
3.4.3	RQ3. Does selecting diverse samples through clustering improve the results?	44
3.5	Threats to Validity	47
3.5.1	Construct Threats	47
3.5.2	Internal Threats	47
3.5.3	Conclusion Threats	48
3.5.4	External Threats	48
3.6	Chapter Summary	48

Table of contents

4 AttSum: A Deep Attention-Based Summarization Model for Bug Report Title Generation	50
4.1 Methodology	51
4.1.1 Framework	51
4.1.2 RoBERTa Encoder	51
4.1.3 Transformers Decoder	54
4.1.4 Copy Mechanism	54
4.2 Experimental Setup	56
4.2.1 Data Preparation	56
4.2.2 Implementation	57
4.2.3 Baselines	57
4.2.4 Evaluation Metrics	58
4.2.5 Statistical Analysis	60
4.3 Evaluation and Analysis	61
4.3.1 RQ1. How does our approach perform under automatic evaluation?	61
4.3.2 RQ2. How does our approach perform under human evaluation?	66
4.3.3 RQ3. How does the size of the training set influence the performance of AttSum?	72
4.3.4 RQ4. How do the feature parts influence the performance of AttSum?	73
4.4 Discussion	75
4.4.1 Automated Metrics for Evaluating Title Quality	75
4.4.2 Is AttSum Good Enough	76
4.4.3 Error Analysis and Suggestions	76
4.5 Threats to Validity	77
4.5.1 Internal Validity	77
4.5.2 External Validity	77
4.6 Chapter Summary	78
5 AdaLog: A Semi-supervised Approach for Industrial Anomaly Detection via Self-adaptive Clustering	79
5.1 Methodology	80
5.1.1 Data Preprocessing	80
5.1.2 Semantic Embedding and Undersampling	82
5.1.3 Self-adaptive Clustering	82

5.1.4	Classification Model Building	87
5.2	Experimental Setup	88
5.2.1	Datasets	88
5.2.2	Baselines	88
5.2.3	Parameter Settings	90
5.2.4	Evaluation Metrics	91
5.3	Results and Analysis	91
5.3.1	Detection Accuracy	92
5.3.2	Clustering Effectiveness	94
5.3.3	Undersampling Effectiveness with Different Ratios	95
5.3.4	Overall Effectiveness with Different Labeled Ratios	99
5.4	Chapter Summary	100
6	On the Influence of Data Resampling for Deep Learning-Based Log Anomaly Detection: Insights and Recommendations	101
6.1	Background	102
6.1.1	Overview of DLLAD Models	102
6.1.2	Existing DLLAD Approaches	104
6.1.3	Data Resampling	104
6.2	Study Design	106
6.2.1	Datasets	106
6.2.2	Evaluation	108
6.2.3	Research Questions	110
6.2.4	Implementations	111
6.3	Results and Analysis	112
6.3.1	RQ1. Do the existing DLLAD approaches perform well enough with varying degrees of class imbalance?	112
6.3.2	RQ2. How does the resampling ratio of normal to abnormal data affect existing DLLAD model performance?	113
6.3.3	RQ3. Does data resampling improve the effectiveness of existing DLLAD approaches?	117
6.4	Discussion	124
6.4.1	Why Do(not) Data Resampling Methods Work	124
6.4.2	Implications of Our Findings	126
6.5	Threats to Validity	127
6.5.1	Limited Models and Datasets	127

Table of contents

6.5.2	Hyperparameter Settings	127
6.5.3	Generalizability	128
6.6	Chapter Summary	128
7	Conclusion and Future Work	129
7.1	Thesis Contributions	129
7.2	Contributions to Software Engineering	131
7.3	Limitations and Future Research Work	132
	References	134

List of figures

1.1	Text-based analysis.	2
1.2	The four text-based analysis tasks in this thesis.	3
3.1	An SBR instance of the <i>Camel</i> project (the text in the orange font represents <i>summary</i> and the text in the grey font represents <i>description</i>).	26
3.2	The framework of CASMS.	28
3.3	The structure of the LSTM cell.	33
3.4	The Boxplot of the g -measure values with CASMS and the three compared methods.	37
3.5	The Boxplot of the pd values with CASMS and the three compared methods.	38
3.6	The Boxplot of the pf values with CASMS and the three compared methods.	38
3.7	Selection of an optimal k^* in the <i>Derby</i> project.	40
3.8	Changes in the best g -measure with the number of NSBRs retained.	42
3.9	The best pd , pf , and g -measure based on two data selection strategies.	45
4.1	The three-phase framework of our approach.	52
4.2	The effect of different ratios of training data on AttSum.	72
5.1	The four-phase framework of AdaLog.	81
5.2	The effect of the undersampling ratio on AdaLog's performance in F1-score. The percentages 0% and 100% indicate the minimum and maximum values of the undersampling range.	96
5.3	The effect of the labeled training data size on AdaLog's performance in F1-score.	98
6.1	The common workflow of DLLAD approaches.	103

List of figures

6.2 The heatmap depicting the statistics of the best performance of the data resampling methods (including OverSampling (OS), UnderSampling (US), and HybridSampling (HS)) with the three resampling ratios (r representing the original ratio of normal to abnormal data) across the three datasets with different window sizes. (a) $ws=20$. (b) $ws=50$. (c) $ws=100$ 115

List of tables

3.1	Details of the collected datasets.	35
3.2	Hyperparameter tuning range for the prediction model.	36
3.3	The evaluation metrics used in our study.	36
3.4	Comparison of the average g -measure, recall, and false alarm of FARSEC, SMOTUNED, LTRWES, and CASMS.	38
3.5	Comparison of the recommended k^* values and the experimental k values yielding the best g -measure.	41
3.6	The NSBR to SBR ratio leading to the best g -measure.	42
3.7	Comparison of the average avg (%), standard deviation sd (%), and median md (%) of g -measure of LTRWES and CASMS.	43
4.1	Automatic evaluation of our approach and baselines. F1, P, and R denote the evaluation metrics for F1-score, Precision, and Recall, respectively.	61
4.2	Automatic evaluation of our approach and baselines on the long bug report dataset.	61
4.3	Statistical analysis on the test set and the long bug report test set. . . .	62
4.4	Examples of automatically generated titles (four approaches) and the original testing titles.	65
4.5	Human evaluation of our approach and baselines.	68
4.6	Human evaluation of our approach and baselines on the long bug report dataset.	69
4.7	Statistical analysis on the 200 samples.	70
4.8	Manual evaluation on bug reports with low-quality titles.	70
4.9	The average Kendall Correlation (τ) and Pearson Correlation (r) among evaluators on the test set.	70
4.10	The average Kendall Correlation (τ) and Pearson Correlation (r) among evaluators on the low-quality test set.	70

List of tables

4.11	Examples of bug reports with low-quality titles.	71
4.12	Ablation studies of AttSum.	73
4.13	Statistical analysis of the effectiveness of the copy mechanism.	74
4.14	Examples of generated titles by AttSum with or without copy mechanism.	74
5.1	An example of log message preprocessing on BGL dataset.	82
5.2	Twelve scenarios of self-adaptive clustering.	84
5.3	Statistics of three datasets.	89
5.4	Performance of different approaches on three datasets. LogRobust and NeuralLog are supervised, while the others are semi-supervised.	92
5.5	Comparison of the effectiveness of different clustering methods on the training set.	93
5.6	Comparison of the effectiveness of different clustering methods on the test set.	94
5.7	Statistics on the undersampling ratio of datasets.	95
5.8	The results of AdaLog without or with undersampling.	97
6.1	Statistics of the three public datasets. TB, <i>Msg</i> , <i>Seq</i> , and <i>A</i> are the abbreviations of Thunderbird, Messages, Sequences, and Anomalies, respectively.	107
6.2	The proportion of abnormal sequences within all log sequences before and after over-/under-/hybrid sampling. r denotes the original ratio of normal to abnormal log sequences in the training dataset, and quarter-based constants are represented as 1/4, 1/2, and 3/4. The last nine columns correspond to the proportion of abnormal sequences after data resampling.	109
6.3	The Recall, Precision, Specificity, F1-score, MCC, and AUC values of the three DLLAD approaches across the three datasets, each with three different window sizes. Bold font highlights the best performance among the three window sizes.	114

6.4	The Recall, Precision, Specificity, F1-score, MCC, and AUC values of CNN when employing various data resampling methods (i.e., NoSampling (NS), SMOTE (SMO), ADASYN (ADA), NearMiss (NM), InstanceHardnessThreshold (IHT), SMOTEENN (SE), SMOTETomek (ST), RandomOverSampling in the feature space (ROS_F), RandomUnderSampling in the feature space (RUS_F), RandomOverSampling applied to raw data (ROS_R), and RandomUnderSampling applied to raw data (RUS_R)) to the three datasets, each with the three different window sizes. Darker cells signify superior performance, while various colors denote statistical significance among data resampling methods for each evaluation metric (determined by the Scott-Knott ESD test with a p -value <0.05), as observed in the subsequent tables.	118
6.5	The Recall, Precision, Specificity, F1-score, MCC, and AUC values of LogRobust when employing various data resampling methods to the three datasets, each with the three different window sizes.	119
6.6	The Recall, Precision, Specificity, F1-score, MCC, and AUC values of NeuralLog when employing various data resampling methods to the three datasets, each with the three different window sizes.	120
6.7	An example of an abnormal log sequence from the Spirit dataset with $ws=20$, where the second log event is highlighted in orange as it represents an anomaly (left), and the top five tokens in the log sequence contributing to the DLLAD model's correct prediction are highlighted in orange (right).	124
6.8	An example of a normal log sequence from the Thunderbird dataset with $ws=20$, along with the top five tokens in the log sequence contributing to the DLLAD model's prediction (right).	125

Chapter 1

Introduction

In the field of intelligent software engineering, the selection of appropriate text representations and representation learning methods is pivotal for effective text-based analysis tasks (presented in Figure 1.1). These chosen techniques play a critical role in capturing and analyzing contextual information within the text, serving as the foundation for bug report analysis [1–7] and log anomaly detection [8–14], both of which primarily involve textual data. The evolution of text representation techniques has been swift [15], advancing from hand-crafted features like Bag of Words (BoW) to embedding vectors such as Latent Semantic Analysis (LSA) [16], Word2vec [17], and Embeddings from Language Models (ELMo) [18]. More recently, text embeddings based on Transformer-based models like Generative Pre-trained Transformer (GPT) [19], Bidirectional Encoder Representations from Transformers (BERT) [20], and their extensions have gained prominence. Furthermore, representation learning methods for these representations are diverse, primarily encompassing machine learning [2, 21] and deep learning [22, 12]. Various text-based analysis tasks necessitate different combinations of text representations and learning methods, occasionally incorporating techniques like clustering and resampling for enhanced representation learning.

For instance, Jiang et al. [5] utilized *word2vec* to represent Non-Security Bug Report (NSBR) and Security Bug Report (SBR) features, employing a ranking model $BM25F_{ext}$ to calculate the similarity between the two classes for SBR prediction. Shi et al. [7] integrated Recurrent Neural Network (RNN)-based Sequene-to-Sequene (Seq2Seq) models with different features and demonstrated superior performance in summarization tasks, such as bug report title generation. Le et al. [12] explored the effectiveness of Deep Learning-based Log Anomaly Detection (DLLAD) approaches and observed that DLLAD models trained on highly

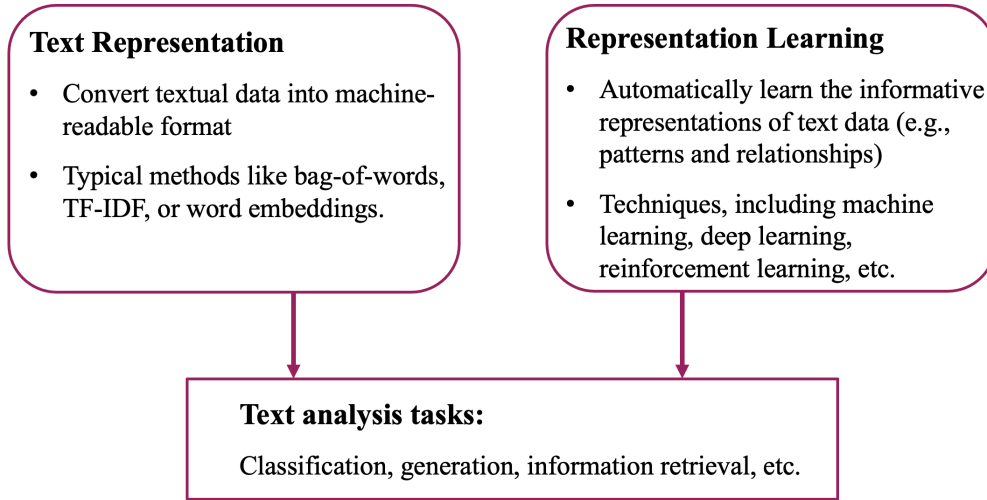


Fig. 1.1 Text-based analysis.

imbalanced datasets yield low precision or recall values. Many DLLAD approaches employed a pre-trained model for text representation and a DL model for log anomaly prediction. For example, Zhang et al. [10] combined a pre-trained *word2vec* model FastText [23] with Term Frequency-Inverse Document Frequency (*tf-idf*) weights for log representation and an attention-based Bidirectional Long Short-Term Memory (BLSTM) model for anomaly detection. Additionally, Le et al. [11] utilized a pre-trained Transformer-based language model BERT [20] for generating representations and applied a transformer encoder for log anomaly detection.

In this thesis, our attention lies on four text-based analysis tasks, as illustrated in Figure 1.2: **(1) security bug report detection**, **(2) bug report title generation**, **(3) log anomaly detection**, and **(4) data resampling in log anomaly detection**. For the first three tasks, we proposed novel approaches to enhance text classification or text generation performance. Regarding log anomaly detection, we introduced a novel DLLAD approach and discovered that integrating random undersampling can mitigate class imbalance and enhance model performance. Furthermore, we conduct a detailed investigation into the effectiveness of data resampling in DLLAD in our subsequent work. These approaches and empirical studies address the underlying research challenges comprehensively.

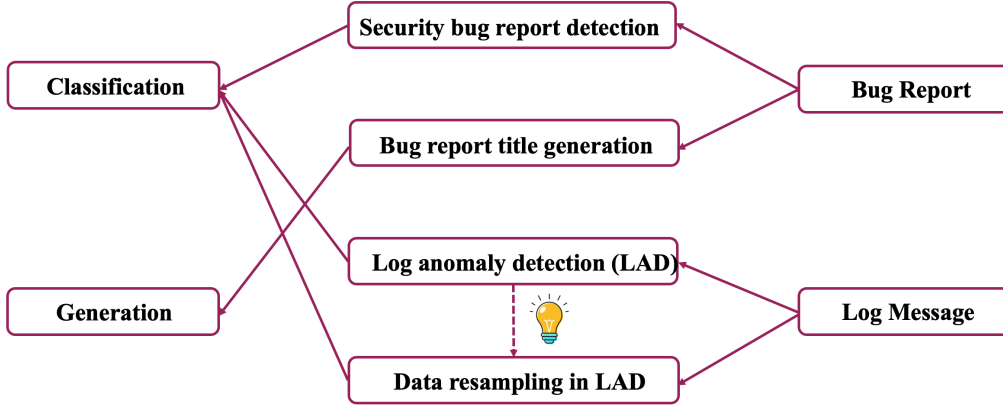


Fig. 1.2 The four text-based analysis tasks in this thesis.

1.1 Security Bug Report Detection

Nowadays, bug tracking systems are widely used to aid developers in software maintenance. Accurately capturing bugs in user-submitted bug reports is crucial for software engineers. These bug reports are typically categorized as SBRs and NSBRs. Priority is given to detecting SBRs as they expose security vulnerabilities that need fixing before disclosure [24]. A published security bug can lead to potential security breaches and significant damage to software products [1]. Text-based prediction methods have been developed to efficiently and accurately identify SBRs. Some approaches [1–3] first represent bug reports as feature vectors by identifying relevant keywords and extracting feature information using techniques like $tf - idf$ [2, 3]. These approaches then utilize classification algorithms such as (e.g., Naïve Bayes and Logistic Regression) to build prediction models based on the feature vectors.

The challenges of identifying SBRs usually include:

(1) How to alleviate the class imbalance problem of the training dataset.

Due to the imbalanced nature of collected datasets, the percentage of SBRs typically ranges from 0.5% to 9% [2, 4, 5]. Prediction models trained on highly imbalanced datasets tend to focus more on NSBRs and misclassify unknown bug reports as NSBRs. However, accurately detecting SBRs is crucial for software engineers. To address the data imbalance issue, recent methods [2, 4, 5] first rank NSBRs using various metrics (e.g., keyword scoring [2, 4] and $BM25F_{ext}$ for similarity calculation [5]). Peters et al. [2] selected the top 100 terms with the highest $tf - idf$ values in SBRs as security-related keywords, noting that 74% of these keywords also appeared in NSBR contexts. However, Jiang et al. [5] argued that many of these security-related keywords may not be related to security. Instead, they ranked NSBRs based on content similarity

rather than individual words, retaining only the most dissimilar NSBRs to SBRs in the feature space.

(2) How to capture the semantic and sequential information of bug reports better. Previous studies [1, 2, 4] adopted term-based approaches (e.g., $tf - idf$) to evaluate the importance of each term in bug reports. These approaches relied on machine learning algorithms to construct classifiers but struggled to capture semantic information and long-term dependencies in bug reports. Our experimental datasets revealed that bug reports contain an average of 75.86 terms, with the longest report having 4,859 terms. This abundance of information requires a more comprehensive approach to model the order and semantic information of terms. In contrast, insufficient information will cause high-dimensional sparsity in the feature space and the loss of temporal relationships between terms.

(3) How to make the approach more efficient and robust. Jiang et al.’s approach [5] achieves good performance through extensive experimentation, conducting a total of 240 experiments on each project. The ratio of NSBRs to SBRs in their best results ranges from 1:1 to 10:1 across 12 combinations of two filters and six classifiers. However, their approach’s robustness evaluation is not ideal, as the g -measure of LTRWES varies significantly with the resampling ratio, resulting in a high standard deviation for its g -measure.

Considering the above challenges, we propose **CASMS** (combining Clustering with Attention Semantic Model for identifying Security bug reports) to efficiently alleviate class imbalance and enhance classification stability. CASMS utilizes the k -means clustering algorithm to group NSBRs, selecting samples closest to cluster centroids to represent diverse NSBRs. This process ensures different potential types of NSBRs are extracted for training. Specifically, we first use the Elbow algorithm to calculate the optimal number of clusters k . The core of security bug report detection is a text-based analysis task. We adopt a Convolutional Neural Networks (CNN)-BLSTM architecture to capture contextual and sequential information within bug reports. First, the CNN layer processes both retained NSBRs and all SBRs to capture the correlation between consecutive terms in the reports. Subsequently, the BLSTM layer extracts correlations between distant terms, followed by the Attention layer, which prioritizes the relevant input parts. This approach contrasts with the general machine learning algorithms utilized in previous works [1, 2, 4, 5], as our prediction model incorporates contextual and sequential information from bug reports.

Despite a slightly higher false alarm rate compared to baselines like FARSEC [2], SMOTUNED [4], and LTRWES [5], CASMS outperforms these baselines in terms

of average g -measure by 4.09% to 24.26% and average recall by 10.33% to 36.24%. The training ratio of NSBRs to SBRs for optimal results ranges from 1:1 to 3:1, with CASMS showing consistently lower standard deviations in g -measure (1.47% to 7.4%) compared to LTRWES. For instance, in the Wicket project, CASMS has a standard deviation of 7.44%, while LTRWES with rs -filter and ms -filter has standard deviations of 11.74% and 15.49%, respectively.

1.2 Bug Report Title Generation

Since bugs are ubiquitous in the software life cycle [25–29], the existence of software bugs encourages bug reporters to record some information about them. However, mainly due to the lack of software management experience and necessary skills, many bug reports provided by reporters suffer from low quality [30]. To improve bug report quality, bug trackers like Jira¹ and Bugzilla² have issued guidelines on crafting constructive bug reports, emphasizing concise and precise bug report titles as a mandatory field³. Despite these guidelines, ad-hoc titles containing scant or noisy information (e.g., brief descriptions, unclear expressions, and irrelevant content) are frequently observed [6].

Chen et al. [6] introduced iTAPE, leveraging a Seq2Seq model to automatically generate bug report titles. To address the lack of suitable data for this task, they curated a dataset with high-quality titles based on three newly proposed heuristic rules. While iTAPE performs well, it faces limitations such as using static pre-trained word vectors like Global Vectors for Word Representation (GloVe) for text representation, leading to ambiguity with polysemous words and imprecise semantics in bug reports. Furthermore, iTAPE’s one-layer LSTM architecture with a copy mechanism faces challenges with long-range dependencies and rare terms [31]. For instance, in their collected dataset, the average sample length is approximately 120 tokens, with the longest reaching 1106 tokens. These extended dependencies between distant terms, coupled with memory constraints, hinder its ability to memorize historical data and infer global dependencies efficiently [31].

Hence, we propose **AttSum**, a deep Attention-based Summarization model, to showcase the impact of semantic and global information from bug report bodies on enhancing bug report title generation performance. AttSum leverages the pre-trained

¹<https://atlassian.design/content/writing-guidelines/empty-state/>

²<https://bugzilla.mozilla.org/page.cgi?id=bug-writing.html>

³<https://testlio.com/blog/the-ideal-bug-report/>

Chapter 1. Introduction

Robustly Optimized BERT approach (RoBERTa) encoder for deep text representation, enabling better capture of contextual semantic information by comprehensively analyzing the entire term sequence at once. Then, AttSum employs a stacked Transformer decoder with multi-head attention to generate bug report titles, effectively handling long-range dependencies between the input and the output sequences regardless of distance. The multi-layered architecture of AttSum facilitates deep-level semantic information capture, enhancing its performance in summarizing bug report bodies. Furthermore, AttSum utilizes a copy mechanism to address rare term challenges, ensuring important but infrequently occurring terms are correctly handled in bug report title generation.

To evaluate our approach AttSum, we use the dataset collected by Chen et al. [6], comprising 333,563 “<body, title>” pairs from 992,730 bug reports on GitHub. We then compare AttSum’s performance against two recent advanced abstractive summarization models (i.e., iTAPE [6] and NATS [7]), as well as an extractive approach (ExtSen), employing two automatic evaluation metrics and human assessment. The experimental results demonstrate that AttSum surpasses three baselines by 3.4%-34.7% (Recall-Oriented Understudy for Gisting Evaluation [32], ROUGE-1), by 24.4%-58.8% (ROUGE-2), by 10.7%-31.0% (ROUGE-L) in terms of F1, and also exhibits superior performance by an average of 7.7%-42.3% in terms of BiLingual Evaluation Understudy (BLEU) [33]. Statistical analysis (including *t*-tests and Cohen’s *d*) indicates that AttSum significantly outperforms the baselines in most cases.

We also invite several qualified volunteers to evaluate the quality of our generated titles from a human perspective. Our approach excels significantly and outperforms the baselines in terms of *Accuracy*, *Comprehensiveness*, and *Lucidity*. Additionally, to improve the quality of initially poor titles, we conduct a practical evaluation using a specific dataset of 50 randomly selected samples, confirming our approach’s capability to produce more reasonable and accurate titles. Moreover, we assess agreement levels using Kendall and Pearson correlations, revealing high consistency in evaluators’ scores. Furthermore, we analyze the impact of training dataset size, demonstrating AttSum’s robustness despite limited data. Ablation studies on the effectiveness of the encoder, decoder, and copy mechanism indicate that AttSum incorporating the copy mechanism achieves superior performance on automatic evaluation metrics. In summary, both automatic evaluations and instance analysis affirm that AttSum excels in generating higher-quality titles for bug reports.

1.3 Log Anomaly Detection

Anomaly detection has gained significant attention in industry due to the need for intelligent data processing and analysis [34–37]. Even minor anomalies in industrial systems can cause issues like data corruption and product performance degradation [11, 12]. Logs have been widely used in reliability enhancement tasks such as system status checks, facility monitoring, and anomaly detection because they record essential system runtime information [38]. This has inspired extensive research into employing machine learning and deep learning techniques for log anomaly detection. Existing approaches are typically categorized into supervised (e.g., LogRobust [10], NeuralLog [11]), semi-supervised (e.g., DeepLog [8], LogAnomaly [9], PLELog [13]), and unsupervised methods (e.g., LogCluster [39]). Supervised approaches often perform better due to their use of labeled data during training. However, they are impractical in industry because they require large labeled datasets and are sensitive to mislabeled data [13, 12]. Unsupervised techniques face challenges due to significant log data instability. Frequent log modifications may result in some incoming log sequences not appearing in the training set [13]. Consequently, effectively deploying unsupervised approaches in the industry can be highly challenging.

Compared to supervised approaches, semi-supervised approaches require only a small amount of labeled data for training, reducing manual labeling costs and reliance on labeled data. Unlike unsupervised approaches, semi-supervised ones leverage part of the training label information, such as feature-label correlations, to optimize training objectives effectively. Although existing semi-supervised approaches achieve decent accuracy on public log datasets (e.g., HDFS [40]), several critical issues remain unresolved, limiting their practical utility. *Firstly*, real-world log data are often imbalanced, with a higher number of normal logs compared to abnormal logs. For example, only 0.82% of the logs in the Thunderbird dataset are anomalous. This imbalance leads to existing approaches often misidentifying anomalies as normal cases. *Secondly*, most semi-supervised approaches rely on clustering algorithms for grouping unlabeled data, which are sensitive to parameter settings and require extensive manual adjustments (e.g., cluster size and minimum samples [13] and similarity threshold [41]). *Thirdly*, the choice of clustering algorithms significantly impacts the performance of the semi-supervised pipeline [13, 41]. Existing clustering algorithms simply predict labels for unlabeled samples without distinguishing different situations (e.g., the correlation between samples in a cluster). This rough labeling often

results in severe performance degradation (e.g., lower precision and specificity) in the subsequent steps due to high false positives (FP).

To address these problems, we propose **AdaLog**, a semi-supervised approach based on self-Adaptive clustering for Log-based anomaly detection. AdaLog employs a pre-trained BERT model [20] for semantic representation and adopts a self-adaptive clustering method based on k -means, dividing label probabilities into twelve situations. Specifically, AdaLog calculates the label probability of each unlabeled sample by considering its distance to the cluster centroid and the distance averages of labeled samples in two classes. This refined calculation is then used for model training. To address dataset imbalance, AdaLog incorporates undersampling before clustering to reduce the imbalanced ratio of normal and abnormal data. Finally, AdaLog employs a Transformer-based model [31] for log anomaly detection.

We conduct comprehensive experiments to evaluate the performance of AdaLog across three widely-used log anomaly datasets (HDFS [40], BGL [42], and Thunderbird [42]). The results showcase that AdaLog significantly outperforms three state-of-the-art (SOTA) semi-supervised approaches, with average F1 improvements of 2489.8% (DeepLog [8]), 2448.9% (LogAnomaly [9]), and 17.8% (PLELog [13]), respectively. Furthermore, AdaLog even surpasses two SOTA supervised approaches in F1 by 23.8% (LogRobust [10]) and 10.9% (NeuralLog [11]). Our ablation studies also highlight the effectiveness of the proposed self-adaptive clustering method and the necessity of undersampling.

1.4 Data Resampling in Log Anomaly Detection

Logs, essential for troubleshooting and performance monitoring [38], are crucial for system maintenance [43–46]. Hence, there’s significant interest in leveraging logs for anomaly detection. Recently, many Deep Learning-based Log Anomaly Detection (DLLAD) approaches [8, 47, 10, 9, 11, 13, 48] have been proposed to automatically identify system anomalies, showing promising results.

In real-world scenarios in DLLAD, the proportion of normal data significantly outweighs that of abnormal data. For instance, consider the Thunderbird dataset, one of the commonly used public datasets where logs are grouped into log sequences (with 20, 50, or 100 logs constituting a sequence) for data analysis. In this dataset, anomalies only account for 0.16%–0.35% of the total, highlighting the serious imbalance in the data distribution. Le et al. [12] have revealed that DLLAD models trained on highly imbalanced datasets exhibit low precision or recall values. Low recall leads to

1.4 Data Resampling in Log Anomaly Detection

missed anomalies, leaving potential threats undetected, while low precision generates numerous false alarms, causing alert fatigue and resource wastage on normal logs [13, 12, 48]. Nonetheless, the issue of class imbalance in DLLAD has remained unaddressed. Data resampling offers an alternative by generating abnormal data or removing normal data, thereby enabling the model to learn from a more balanced representation of both classes.

In this study, we aim to assess the impact of data resampling on the performance of existing DLLAD approaches and explore the optimal ratio of normal to abnormal data for data resampling. To achieve these goals, we conduct an extensive empirical study by employing three oversampling methods (*Random OverSampling (ROS)*, *SMOTE* [49], and *ADASYN* [50]), three undersampling methods (*Random UnderSampling (RUS)*, *NearMiss* [51], and *InstanceHardnessThreshold* [52]), and two hybrid sampling methods (*SMOTEENN* [53] and *SMOTETomek* [53]) to three supervised DLLAD approaches (CNN [47], LogRobust [10], NeuralLog [11]) across three publicly available datasets. We compare the results with DLLAD approaches with *NoSampling* (using the original dataset). Furthermore, the data resampling methods can also be categorized into resampling on raw data and resampling in the feature space. It is important to note that many data resampling methods are designed for application only within the feature space, as they rely on distance computations. Simpler methods, like *ROS* and *RUS*, can be applied to both raw data (duplicating/removing log sequences with identical texts) and feature space (duplicating/removing sequences with the same embedding vectors). We structure our study with the following research questions:

(1) Do the existing DLLAD approaches perform well enough with varying degrees of class imbalance? We first evaluate the performance of existing DLLAD approaches across datasets with diverse levels of class imbalance. **Findings:** The performance of DLLAD approaches is greatly affected by the degree of class imbalance, with their effectiveness notably decreasing when faced with severe data imbalance.

(2) How does the resampling ratio of normal to abnormal data affect the ability of data resampling? We explore how varying the resampling ratio of normal to abnormal data impacts the results by using quarter-based multipliers of the original ratio of normal to abnormal log sequences. **Findings:** The effectiveness of oversampling methods on DLLAD approaches is maximized when generating more abnormal log sequences. Conversely, removing fewer normal log sequences enhances the effectiveness of undersampling methods on DLLAD approaches. When DLLAD

approaches already perform well on specific datasets without applying data resampling methods, they become less sensitive to the choice of the resampling ratio.

(3) Does data resampling improve the effectiveness of existing DLLAD approaches? We assess the effectiveness of data resampling on DLLAD approaches utilizing an optimal resampling ratio (obtained from RQ2) of normal to abnormal data. **Findings:** Overall, oversampling methods demonstrate superior performance compared to undersampling and hybrid sampling methods. Remarkably, the straightforward methods applied directly to raw data outperform other methods applied within the feature space. Surprisingly, in many scenarios, some more advanced undersampling methods (i.e., *NearMiss* and *InstanceHardnessThreshold*), and even a hybrid sampling method *SMOTEENN* aimed at mitigating data imbalance, fail to effectively enhance the performance of DLLAD approaches.

1.5 Contributions of Thesis

In this thesis, three novel deep learning-based approaches (CASMS, AttSum, AdaLog) are proposed, accompanied by one comprehensive empirical study addressing persistent research challenges.

- In the domain of security bug report detection, we propose CASMS (combining Clustering with the Attention Semantic Model for Security bug report detection) to automatically select diverse NSBRs and learn semantic/sequential information from bug reports, enhancing performance.
- We evaluate CASMS on five real-world projects, showcasing its efficiency and robustness without much tuning effort. It automatically determines the optimal number of clusters and adjusts the ratio scope of NSBRs to SBRs.
- For bug report title generation, we introduce AttSum, a deep Attention-based Summarization approach that utilizes the pre-trained RoBERTa encoder to comprehensively represent the terms in report bodies, a stacked Transformer decoder to address long-range dependencies, and a copy mechanism to solve the rare term issue.
- The experimental results demonstrate that AttSum performs better than the baselines on both automatic and manual evaluation; It not only demonstrates

robustness but also improves the quality of low-quality titles. We open source the code of our approach⁴ to facilitate further studies.

- In Log Anomaly Detection (LAD), we present AdaLog, a semi-supervised approach designed to address the primary concerns of existing semi-supervised approaches. At the heart of AdaLog lies a self-adaptive clustering method featuring twelve carefully crafted situations, enhancing label probability calculation and representation learning.
- The experimental results show that AdaLog surpasses SOTA semi-supervised approaches by 17.8%–2489.8% in F1, and it also outperforms two supervised approaches by 10.9%–23.8%. Our implementation is publicly accessible⁵.
- Furthermore, we delve into the effectiveness of data resampling methods in Deep Learning-based Log Anomaly Detection (DLLAD). Our study represents the first comprehensive assessment of how different data resampling methods affect model performance in DLLAD. We conduct a total of 4,185 experiments, employing ten data resampling methods in conjunction with existing DLLAD approaches. Our study includes a thorough evaluation and statistical analysis across three publicly available benchmark datasets.
- Based on the empirical results, we conclude the findings and provide recommendations to researchers and software developers in LAD. We recommend that practitioners 1) prefer oversampling with generating more abnormal log sequences over undersampling and hybrid sampling and 2) prioritize data resampling on raw data over data resampling within the feature space. However, we do not recommend *SMOTEENN* for datasets with extremely high class imbalance. We open source our replication package⁶ for further research.

1.6 Organization of the Thesis

The remainder of this thesis is structured as follows:

⁴<https://github.com/mkkmaomao/AttSum>

⁵<https://github.com/mkkmaomao/AdaLog>

⁶<https://github.com/ResamplingDLLAD/ResamplingEmpirical>

Chapter 1. Introduction

Chapter 2 provides a comprehensive review of related work in the thesis, covering topics such as security bug report detection, text summarization, log anomaly detection, and data resampling.

Chapter 3 introduces CASMS, a novel approach that utilizes clustering to select diverse non-security bug reports to address class imbalance. An attention-based CNN-BLSTM model is employed to capture semantic/sequential information for effective security bug report detection. Extensive experiments demonstrate the superior performance of CASMS compared to baselines.

Chapter 4 presents AttSum, an attention-based summarization approach using an Encoder-Decoder framework to generate high-quality bug report titles by representing comprehensive information from report bodies. AttSum is evaluated on real-world bug report datasets and outperforms baselines.

Chapter 5 describes AdaLog, which employs a self-adaptive clustering method to pseudo-label unlabeled data before training. Experimental results on publicly available datasets showcase the effectiveness of AdaLog.

Chapter 6 explores the impact of data resampling methods on deep learning-based log anomaly detection approaches. Through comprehensive analysis, this chapter provides insights and recommendations for researchers in the field.

Finally, Chapter 7 concludes the thesis by summarizing the research findings, highlighting the main contributions, and suggesting avenues for future work.

Chapter 2

Literature Review

In this chapter, we delve into a range of text-based analysis tasks pertinent to this thesis, such as security bug report detection, bug report title generation, log anomaly detection, and data resampling.

2.1 Security Bug Report Detection

In the early phase, bug tracking systems are used to help engineers to identify duplicate bug reports [54–58], evaluate the severity or priority of bug reports [59–61], trace bug reports back to relevant source documents [62–66], analyze and predict the effort needed to fix software bugs [67], work on characteristics of software vulnerabilities [68, 69], and evaluate the ability of code analysis tools to detect security vulnerabilities [70]. Most of these approaches applied textual similarity metrics (e.g., cosine similarity) and machine learning methods (e.g., SVM and KNN) for extracting textual information, yet they often overlook sequential and semantic information. In addition, studies specifically focusing on security bug report detection [1–5] are limited, predominantly relying on machine learning methods for model training. Consequently, the integration of sequential and semantic information remains largely unexplored in prior work.

Gegick et al. [1] first highlighted the issue of bug reporters mistakenly labeling security bug reports (SBRs) as non-security bug reports (NSBRs) due to a lack of expertise in the security domain. To address this, they employed text mining techniques on bug report descriptions to create term-by-document frequency matrices, upon which they trained a statistical model. Their empirical findings revealed a significant proportion (78%) of SBRs being inaccurately labeled as NSBRs within a large Cisco

Chapter 2. Literature Review

software system. This suggests that the features extracted during training, notably the security-related keywords, could contribute to misclassification of bug reports.

In a similar vein, Peters et al. [2] discovered that at least 74% of security-related keywords present in both SBRs and NSBRs, termed as security cross words, could lead to mislabeling of SBRs. Consequently, they proposed the FARSEC framework to identify SBRs by filtering out NSBRs containing security cross words. Initially, they computed the $tf - idf$ value for each term and selected the top-100 terms with the highest $tf - idf$ values as security-related keywords. Subsequently, these terms were utilized to construct term-document matrices (referred to as the feature set). Then, each term in the training set was scored, with terms from the feature set given higher weights through various filters. Finally, bug reports were ranked, and NSBRs scoring above a certain threshold were filtered out. Additionally, they incorporated the Closest List Noise Identification (CLNI) algorithm [71], which is based on Euclidean Distance, for prediction purposes. Goseva et al. [3] introduced both supervised and unsupervised approaches for identifying security bug reports. In both methodologies, they utilized three feature extraction methods: binary bag-of-words frequency (bf), term frequency (tf), and term frequency-inverse document frequency ($tf - idf$). For the supervised approach, they integrated machine learning algorithms with feature vectors to classify bug reports. Their unique contribution lies in exploring different training set sizes to ascertain the minimum size necessary for achieving satisfactory classification results. Additionally, they proposed an unsupervised approach based on anomaly detection principles to identify security bug reports. Evaluation of these approaches was conducted using data from two NASA missions obtained from issue tracking systems.

Building upon the FARSEC framework, Shu et al. [4] incorporated the Synthetic Minority Over-sampling Technique (SMOTE) algorithm [49] along with FARSEC filters for data preprocessing. They employed the Differential Evolution (DE) algorithm [72, 73] to optimize the parameters of SMOTE and the machine learning classifiers independently. Through experimentation, they recommended utilizing DE exclusively for tuning SMOTE, resulting in SMOTUNED, rather than for tuning the learners due to its time-consuming nature. Consequently, they observed a substantial enhancement in recall (pd) of prediction models with a moderate increase in false alarm rate (pf), leading to SMOTUNED outperforming FARSEC in terms of the g -measure.

In a recent study, Jiang et al. [5] introduced LTRWES, a novel approach distinct from keyword-based FARSEC. LTRWES employed a content-based data filtering strategy that considered the entire content of bug reports rather than focusing solely

on individual keywords. Initially, LTRWES calculated the content similarity between each NSBR and all SBRs using a ranking model $BM25F_{ext}$ [74]. Subsequently, it extracted the top- m NSBRs that exhibited the greatest dissimilarity to SBRs. The underlying concept of LTRWES mirrors that of FARSEC, aiming to retain the most dissimilar NSBRs to SBRs to achieve higher recall. To mitigate issues related to vector sparsity and capture semantic information, LTRWES also utilized *Word2vec* to transform the remaining NSBRs, along with SBRs, into low-dimensional vectors prior to training.

However, there are several issues with the previous approaches: (1) These approaches focused on retaining the most dissimilar NSBRs to SBRs rather than ensuring diversity among NSBRs. (2) They relied on machine learning techniques for model training, neglecting the semantic and sequential information inherent in bug reports. (3) Despite achieving good performance, LTRWES exhibited a lack of robustness, even after conducting repeated experiments. For instance, when utilizing a combination of the SVM classifier and the *ms* filter, the authors experimented with different ratios of NSBRs to SBRs for training, ranging from 1:1 to 10:1, in pursuit of optimal prediction results. However, the number of remaining training NSBRs varied widely, from 4 to 770. Additionally, the *g*-measure of LTRWES exhibited significant variability with changes in the resampling ratio.

2.2 Text Summarization

For text summarization tasks, people receive a tremendous amount of information from multiple online platforms and web pages. However, it is not such easy to quickly understand the core point from acres of text within a limited time, which promotes research in automatic text summarization to generate readable and informative summaries with main ideas [75]. In general, extractive and abstractive ways are normally used for automatic summarization. The former aims to identify and directly extract the important parts from the source text. In contrast, the latter generates each term from a given vocabulary, which conveys the informative terms and restructures them into a completely shorter text. The fundamental role of bug report title generation is to help software bug reporters write a high-quality title before submitting bug reports. Since the generated title is expected to contain the necessary terms and can accurately summarize the content of the report body, this bug report title generation task can be treated as a one-sentence text summarization task.

2.2.1 Extractive Methods

A framework consisting of a hierarchical encoder and an attention-based extractor for document summarization was developed by Cheng et al. [76]. Then many researchers proposed enhanced models to adapt to different problems. Typically, extractive models adopt a binary classifier to judge whether the sentence is included in the extractive summary by learning the labels of sentences (i.e., extraction or not). SummaRuNNer as an RNN-based labeling sequence model was proposed by Nallapati et al. [77] for extractive summarization that can be trained in an end-to-end manner. They assigned each sentence a probability of being extracted and then selected sentences based on it. Zhou et al. [78] integrated the two steps (i.e., scoring and selection) to predict the relative importance of the rest sentences given the previous extraction. Narayan et al. [79] selected sentences as the extracted text of the summary by ranking, and applied reinforcement learning to globally optimize the ROUGE for the summarization task. Compared to the models that selecting sentences relies on their binary labels, Zhang et al. [80] proposed a latent variable model LATENT, where they regarded sentences as latent variables and used them to infer the golden summaries instead of the labels. BANDITSUM was put forward by Dong et al. [81], which combined neural extractive summarizers with reinforcement learning. It also removed the sequential binary labeling setting and treated summaries as a contextual bandit. However, sometimes the contextual features are unable to be learned well, Liu et al. [82] utilized pre-training BERT[83] with a powerful architecture for vector representation and built summarization layers (e.g., linear layer) stacked on the vectors for fine-tuning.

The above models aim to generate a summary containing many sentences and indeed have a good performance on semantic parsing. Nevertheless, our bug report title generation task requires generating a title (i.e., one sentence) that effectively summarizes the report body in very few words. Thus, it is not expected that extractive methods can directly extract one sentence from a descriptive text up to 1106 tokens in this study.

2.2.2 Abstractive Methods

In recent years, the emergence of Encoder-Decoder models for some tasks (e.g., machine translation [84–87], speech recognition [88, 89], video captions [90]) dramatically promoted the development of abstractive techniques. Rush et al. [91] adopted convolutional models as encoder and an attentional feed-forward neural network for summary generation, which was referred by Nallapati et al.[92] who then

proposed an attentional Encoder-Decoder RNN model for text summarization. An open-source toolkit for neural machine translations proposed by Klein et al. [22] was made public, which is based on an attentional Seq2Seq model and is widely used in many Natural Language Processing (NLP) tasks. Shi et al. [7] developed another toolkit named Neural Abstractive Text Summarizer (NATS) for abstractive summarization. It combines typical Seq2Seq models with many features and is possibly applied in different datasets.

Consequently, Transformer-based Seq2Seq models (e.g., BERT [83], UNILM [93], BART [94], RoBERTa [95], PEGASUS [96]) for pre-training have been extended to abstractive summarization. For example, Liu et al. [97] introduced BERTSUM based on BERT as the encoder and stacked several Transformer layers as the extractive model for abstractive summarization. Inspired by these studies, we use a pre-training model RoBERTa for text representation and a stacked Transformer decoder for text generation to construct our Encoder-Decoder model. Due to the presence of rare tokens in bug reports, we also incorporate a copy mechanism in our approach.

2.3 Log Anomaly Detection

In the realm of Log Anomaly Detection (LAD), supervised approaches often outperform semi-supervised and unsupervised approaches owing to their utilization of a large pool of labeled data for training. However, semi-supervised and unsupervised approaches are normally considered more practical due to their diminished reliance on labeled data.

2.3.1 Supervised Approaches

Numerous machine learning-based approaches have been proposed for log anomaly detection. Liang et al. [98] introduced four classifiers for predicting failure log events. Chen et al. [99] employed a decision tree model to classify logs within eBay’s web request logging system, while Farshchi et al. [100] presented a regression-based approach for detecting application operation failures. Subsequently, a plethora of deep learning-based approaches emerged for log anomaly detection (DLLAD).

Zhang et al. [101] combined log template extraction with $tf-idf$ to represent log templates as vectors, leveraging an LSTM model for anomaly prediction. Similarly, Vinayakumar et al. [102] utilized a stacked-LSTM model to learn temporal patterns using sparse representations. Lu et al. [47] employed CNN with three filters to learn

Chapter 2. Literature Review

local semantic information from log data. Zhang et al. [10] proposed LogRobust, integrating attention mechanisms with a Bi-LSTM model to capture comprehensive log sequences in both directions. Li et al. [103] utilized BERT for semantic representation and an attention-based BLSTM for anomaly detection. Zhang et al. [104] proposed DeepTraLog, a deep learning-based microservice anomaly detection approach that utilizes a unified graph representation to describe the complex structure of traces together with log data, and applies a gated graph neural networks (GGNNs) based deep SVDD model to detect anomalies. Hashemi et al. [105] proposed a training pair generation algorithm to train the siamese network combined with LSTM and GRU layers to reduce training overhead while maintaining effectiveness. Fu et al. [106] employed a hybrid deep neural network consisting of a Transformer encoder with IDF-weights for semantic representation and a deep neural network combining the Mogrifier LSTM with CNN to capture global and local log patterns simultaneously. Hashemi et al. [107] adopted a single deep neural network, HierarchicalCNN, for end-to-end learning, containing multiple convolutions and fully-connected layers. Zhang et al. [108] proposed MetaLog, which incorporates a globally consistent semantic embedding module using GloVe and leverages the meta-learning paradigm based on GRU to improve the model's generalization ability. Guo et al. [109] proposed LogFormer, a unified Transformer-based framework with a two-stage process, pre-training the log-attention encoder on the source domain to obtain shared semantic log information, followed by fine-tuning on the target domain using a plug-in mechanism as an adapter.

Furthermore, several works have developed frameworks for log anomaly detection based on pre-trained representation models and Transformer-based classification models. Le et al. [11] introduced NeuralLog, leveraging BERT for embedding representation and a Transformer encoder for log anomaly detection. Xie et al. [110] described LogRep, which captures semantic and numeric information using a BERT-based semantic representation module and a position-aware numeric representation module, followed by anomaly detection with a Transformer-based module. He et al. [111] developed LogBP-LORA, a parameter-efficient log anomaly detection framework that combines BERT and Low-Rank Adaptation (LORA) by adding a bypass for the self-attention layer to reduce training overhead. Wang et al. [112] employed SIMCSE-BERT for semantic representation and a Transformer-based classification model combined with an LSTM model for numerical key-value pair learning.

2.3.2 Semi-Supervised and Unsupervised Approaches

DeepLog [8] utilized an LSTM model to autonomously learn log patterns from normal log sequences, detecting anomalies when incoming log sequences deviate from normal execution. LogAnomaly [9] considered semantic information by matching log sequences against generated templates to learn normal patterns. PLELog [13] combined HDBSCAN clustering for probabilistic label estimation with an attention-based GRU model for log sequence classification. LogOnline [113] was designed to continuously learn normal sequence patterns as new log sequences appear. It used $tf - idf$ -weighted FastText to transform logs into semantic vectors, then employed LSTM layers and a fully connected neural network to calculate the combined loss for anomaly detection. Hades [14] combined both log and trace data for anomaly detection, adopting FastText and Transformer to learn log representations and employing a hierarchical encoder with a causal convolution network for metric representations. The semi-supervised training involved initially training with a few labeled data points, then pseudolabeling the remaining unlabeled data and updating the model iteratively. LogLoader [114] utilized a Transformer encoder to capture normal patterns from log data. It identified anomalies by predicting a set of candidate log templates for the next log message under normal conditions, flagging an anomaly if the actual log message template did not match any candidates.

LogCluster [39], an unsupervised approach, grouped log sequences via hierarchical clustering based on log event weights. LAnoBERT [115] leveraged BERT to learn log information without parsing. Yang et al. [116] enhanced the traditional unsupervised PCA (Principal Component Analysis) technique by incorporating a lightweight semantic-based log representation ($tf - idf$ -weighted GloVe) to address the problem of unseen log events, improving anomaly detection effectiveness.

While unsupervised approaches typically underperform compared to semi-supervised ones due to the lack of labeled data, semi-supervised approaches benefit from using a portion of labeled data for training, balancing the need for labeled data with the practical constraints of acquiring it. However, many semi-supervised approaches, particularly those using clustering algorithms, face challenges due to their sensitivity to parameter settings, requiring extensive manual tuning. The choice of clustering algorithm significantly impacts the performance of semi-supervised approaches, potentially leading to unsatisfactory results if not properly configured [13, 41].

2.4 Data Resampling

2.4.1 Data Resampling in Software Engineering

Data resampling has been widely applied to address the class imbalance issue in the field of software engineering, such as quality prediction [117], bug classification [118, 119], defect prediction [120–127], and code smell detection [128, 129]. For example, Zheng et al. [118] analyzed the impact of six data resampling methods (e.g., SMOTE, Mahakil [130], and Rose [131]) on multiple classifiers for bug report classification. They found that the combination of Rose with random forest yielded the best performance. Bennin et al. [123] observed that while their investigated data resampling methods have no statistically significant effect on defect prioritization, these methods improve the defect classification performance with regard to Recall and G-mean. Subsequently, they [125] demonstrated that random undersampling and borderline-SMOTE are the more stable data resampling methods in software defect prediction. Li et al. [129] investigated the effects of 31 imbalanced learning methods on machine learning classifiers for code smell detection. Their study revealed varied impacts of these methods across different code smells, with deep forest consistently enhancing performance. Additionally, certain data resampling methods such as CNN, ENN, BSMOTE, and ROS showed superior performance compared to SMOTE.

Differently from prior work, in the context of log anomaly detection, which poses unique challenges such as the need for careful selection of window sizes, our research conducts an extensive analysis to explore the influence of data resampling methods on model performance across various window sizes. Furthermore, this thesis delves into the effects of some data resampling methods on both raw data and the feature space, providing valuable insights for practitioners in this field.

2.4.2 Data Resampling for Anomaly Detection

Deep learning-based anomaly detection is applicable not only in software log anomaly detection (LAD) but also in diverse fields like fraud detection [132–135], medical diagnosis [136–139], manufacturing defect detection [140], and network intrusion detection [141]. While the workflow for anomaly detection in these domains shares similarities with that of log anomaly detection, specific data preprocessing steps, such as log parsing and grouping, may not be applicable. In these fields, class imbalance also presents a significant challenge that can affect the performance of anomaly detection models. Various data resampling methods have been employed to

address this issue. For example, Heryadi et al. [132] utilized RUS on non-fraudulent transactions and adopted a hybrid CNN-LSTM model to capture both short and long-term financial transaction features. Similarly, Roy et al. [133] implemented RUS with a recommended resampling ratio [142] of 10:1 for non-fraudulent to fraudulent credit card transaction data, followed by an LSTM model for detecting fraud in credit card transactions. In medical anomaly detection, imbalanced datasets pose challenges, leading to sensitivity issues during supervised training [139]. Common strategies, such as data resampling and cost-sensitive training [137], are employed to mitigate these challenges. Li et al. [138] employed a CNN-based model with adjusted class weights to analyze phonocardiograms for abnormal heart sound detection. In network intrusion detection, Abdelkhalek et al. [141] proposed a data resampling approach combining ADASYN and Tomek Links in conjunction with diverse deep learning models (such as LSTM and CNN) for improved detection of malicious attacks, aiming to address the class imbalance issue between normal traffic and attack samples. In summary, the aforementioned studies suggest that certain data resampling methods can enhance model performance, aligning with our findings in this thesis.

However, empirical research regarding the most effective data resampling methods, optimal resampling ratio, and their impacts on both raw data and the feature space is currently lacking in these fields.

2.5 Summary

In this chapter, we have presented the techniques most relevant to this thesis. For each subsection, we provide a summary of recent methodologies and highlight their limitations.

Chapter 3

CASMS: Combining Clustering with Attention Semantic Model for Identifying Security Bug Reports

Detecting security bug reports is crucial for assisting software developers in maintaining software. However, due to class imbalance, prevailing methods typically rely on keyword-based or similarity-based approaches to select NSBRs. They employ machine learning algorithms for constructing classifiers but overlook sequential information analysis and often necessitate extensive experiments to achieve satisfactory performance, lacking robustness.

To overcome these challenges, we propose an innovative approach that combines clustering with an attention-based sequential model to enhance representation learning. Specifically, we utilize $tf - idf$ -weighted *word2vec* for text representation, employ k -means combined with the Elbow method to extract diverse NSBRs, and utilize a CNN-BLSTM model for representation learning during training and prediction.

To validate our approach, we applied it to 44,940 bug reports from five public datasets. Experimental results showcase its superiority over three baselines (FARSEC, SMOTUNED, and LTRWES) in overall performance assessment (g -measure) and correctly identifying SBRs (*recall*), with enhancements ranging from 4.09% to 24.26% and 10.33% to 36.24%, respectively. The approach performs well within the limited ratio ranges of NSBRs to SBRs (1:1 to 3:1), with approximately 20 experiments conducted for each project. Furthermore, by evaluating the robustness of CASMS using the standard deviation indicator, it proves to be more stable than LTRWES. This work was published in the Information and Software Technology journal in 2022 [143].

The subsequent sections of this chapter are organized as follows: Section 3.1 provides background and theoretical foundations, while Section 3.2 introduces CASMS and its methodology. Sections 3.3 and 3.4 delve into the experimental setup and results, respectively. Section 3.5 elaborates on the threats to the validity of this work, and a summary is provided in Section 3.6.

3.1 Background

3.1.1 Text Representation

Text classification typically involves identifying the category of a new document based on probabilities derived from a designated training corpus with labeled data [144]. Adopting appropriate techniques for text representation is essential to interpret the text effectively and assist subsequent representation learning in categorizing data into distinct categories.

Term Frequency-Inverse Document Frequency ($tf - idf$) is one of the most popular term-weighting schemes in text analysis and data mining [145, 146], applicable in various contexts such as hidden impact bug identification [147] and SBRs detection [2, 3]. In the field of information retrieval, $tf - idf$ is represented as a numerical statistic value reflecting the significance of a term in a document corpus or collection. Terms with higher $tf - idf$ values indicate a powerful ability to distinguish the categories of documents and have a stronger relationship with the documents where these terms appear [144]. Given a document collection D , a term t , and an individual document d that belongs to D . The calculation of $tf - idf$ is as following [145]:

$$t_d = f_{t,d} * \log(|D|/f_{t,D}), \quad (3.1)$$

where $f_{t,d}$ is the number of times that term t appears in the document d , $|D|$ is the size of the corpus, and $f_{t,D}$ is the number of documents in which the term t appears in D . Although $tf - idf$ supposes each term in bug reports is independent, it reflects the importance of the term to bug reports where the term exists to a certain extent. Thus, we apply $tf - idf$ to calculate the weight of terms instead of directly generating word vectors in our proposed approach.

Word embedding is critical to construct deep learning models in many NLP tasks [148–155]. In the early days, the original objective of word embedding was to convert text into a statistical representation, such as the one-hot representation. For each word, its corresponding position was set to 1, and the other positions were all 0. However,

the earliest sparse-populated vector was very verbose because it set the dimension of the word vector to be the size of the entire vocabulary, which may cause a dimensional disaster. In addition, the contextual information of words was ignored. To address this problem, Mikolov et al. [156, 17, 157] proposed two language learning algorithms, Continuous Bag-of-Words (CBOW) and Skip-Gram based on *word2vec*, to represent the meaning of a word with respect to other words. The former algorithm predicts the surrounding words according to the given current word, while the latter algorithm predicts the current words based on the distributed representation of the context [144]. Generally, owing to the higher calculation complexity of the Skip-Gram model, CBOW is more suitable for scenarios with small datasets [5, 156]. LTRWES, an approach for security bug report detection, utilizes CBOW to transform the bug reports into low-dimensional vectors for subsequent training. Building upon this method, we multiply the word vectors by the previously calculated word weights to generate the feature vectors for our datasets.

3.1.2 Clustering

Clustering automatically aggregates similar objects into clusters based on shared characteristics. By minimizing variance within clusters, it effectively organizes entities. This technique has been extensively explored in machine learning, including applications in feature selection [158, 159], distance algorithms [160], and grouping methods [161]. In the context of identifying diverse NSBRs, clustering algorithms segment data into distinct categories based on data features.

Overall, there are several types of clustering methods to consider: 1) *Hierarchical-based* methods [162], like CHAMELEON, are known for causing high time complexity [163, 164], making them unsuitable for our study due to time constraints. 2) *Density-based* methods, such as DBSCAN, can handle clusters of arbitrary shapes but are often more computationally intensive and sensitive to parameter settings [165, 166]. This sensitivity, along with challenges in managing clusters with varying sparsity [166], makes them less effective for large-scale, high-dimensional data like bug reports. 3) *Grid-based* methods [167, 168], like CLIQUE, offer speed but are sensitive to parameter selection [169]. Threshold selection, in particular, poses challenges as high thresholds may overlook clusters while low thresholds may merge clusters inappropriately. Additionally, grid-based methods struggle with irregularly distributed data and are susceptible to the curse of dimensionality. 4) *Partition-based* methods [170], although straightforward and

efficient [163, 164, 171], have drawbacks. They may result in locally optimal solutions, and the number of clusters must be predetermined. However, this limitation can be mitigated by conducting multiple trials and employing the Elbow principle for cluster number determination [172, 173].

Given these considerations, we chose the widely-used partition-based method, k -means [174], as it provides a practical balance between performance and efficiency. In k -means, initial centroids are constructed based on incoming data and the specified k value. These centroids are iteratively adjusted based on the distances between centroids and data samples. The resulting centroids and the distances between each sample and its centroid are retained for further analysis.

Before systematically exploring multiple k values, we can roughly estimate the recommended k^* using the Elbow method. In k -means clustering, the objective function is to minimize the square error between samples and centroids. The sum of squared distance errors between each cluster centroid and its member points is termed as distortion (or distortions). Lower distortion indicates tighter clustering, while higher distortion suggests a looser clustering structure. As the number of clusters k increases, distortion typically decreases. However, in datasets with inherent structure, distortion improves until a critical point is reached, after which it diminishes slowly. This critical value k^* can be identified as the optimal cluster number.

3.1.3 Models for Representation Learning

Machine learning algorithms are extensively employed for learning text representations, and recent studies [2, 4, 3, 5] have utilized these algorithms to construct SBR prediction models. These include Naïve Bayes (NB) [175, 176], Logistic Regression (LR) [177, 178], Support Vector Machine (SVM) [179], Random Forest (RF) [180], Multilayer Perceptron (MP) [181], and K Nearest Neighbor (KNN) [182]. However, these algorithms fail to retain information from previous terms continuously; instead, they discard previous information and restart calculations. In essence, they are unable to capture the sequential information necessary to process subsequent data.

Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) are two prominent architectures for modeling and comprehending natural language. CNN excels at learning local patterns from temporal data, while RNN, with its ability to retain information over time, is adept at sequential modeling [183]. RNN [184] is designed to capture long-term dependency within the sequential data, which adopts

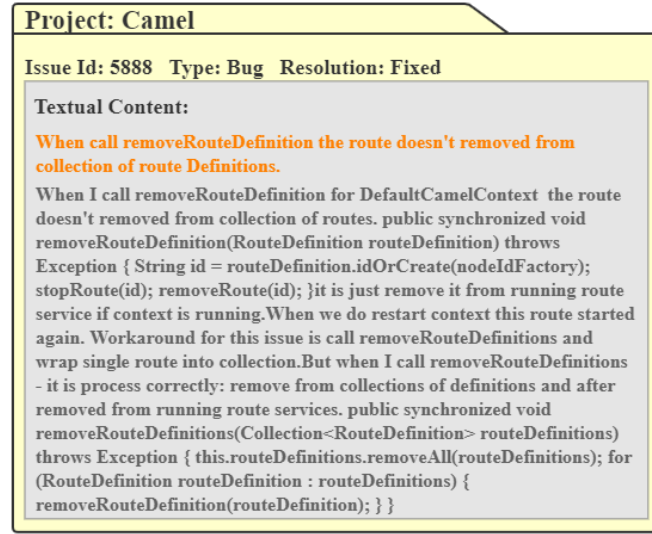


Fig. 3.1 An SBR instance of the *Camel* project (the text in the orange font represents *summary* and the text in the grey font represents *description*).

a simple mechanism of recurrent feedback. In the field of information retrieval, term dependencies [185] are usually defined as statistical co-incidence of terms at the document level. Long-term dependencies refer to relatively long intervals of relevant information and locations. Long Short-Term Memory (LSTM) architecture is derived from RNN. It has recursive connections so that the previous activation state of neurons from previous time steps is used as the context for forming the output. Bidirectional LSTM (BLSTM) was initially proposed by Graves [186] to avoid gradient exploding and vanishing, and it had been applied in many speech recognition tasks [186, 187]. Hence, BLSTM focuses on extracting the correlation between long-distance terms but insufficiently captures the correlation between consecutive terms. For this reason, a mixed model CNN-BLSTM was designed for extracting features and was broadly used in multiple fields [188–191]. The attention mechanism, introduced by the Bengio team [192], allocates weights to terms, enhancing the model's ability to identify impactful components within input sentences. This mechanism has found broad applications in translation, image captioning, and speech recognition [192–194]. The attention mechanism originates from information retrieval systems, where users input a query (Q) and the search engine matches keywords (K). Based on their similarity, the matching content (V) is returned.

In our study, bug reports typically consist of a *summary* and a detailed *description* (as illustrated in Figure 3.1). In our datasets, a bug report contains an average of 75.86

terms, a median of 44 terms, and the longest one contains 4,859 terms. Considering Khandelwal et al.’s observation that LSTM models can effectively handle context sizes of up to 200 tokens on average [195], the LSTM architecture is suitable for capturing long-term dependencies between terms in our datasets, as both the average and median lengths are below 200 terms. This makes CNN-BLSTM a suitable choice for our prediction model. Compared to Transformer models that utilize multi-head attention mechanisms [31], CNN-LSTM is more appropriate for our dataset due to the relatively small data size (refer to Table 3.1). The advantages of CNN-LSTM include 1) delivering competitive results without needing extensive training data, unlike transformers that often require large-scale datasets; 2) demanding less computational power and training time; and 3) being sufficient for binary classification tasks, particularly when compared to the ML classifiers used in our baselines. To further enhance our approach, we integrate a self-attention mechanism into the prediction model, allowing it to automatically identify critical terms for classification, thereby improving the model’s ability to accurately capture important features.

3.2 Methodology

3.2.1 Framework Overview

The CASMS framework, depicted in Figure 3.2, comprises three stages. In the first stage (Data Preparation), CASMS transforms bug reports into weighted word embeddings using a combination of $tf-idf$ and CBOW techniques. In the second stage (Data Filtering), CASMS efficiently extracts diverse NSBRs by employing the k -means clustering algorithm in conjunction with the Elbow method. In the third stage (Training and Prediction), CASMS trains an effective Attention CNN-BLSTM model utilizing the selected NSBRs alongside all SBRs. The intricate processes are elaborated in the subsequent subsections.

3.2.2 Data Preparation

Text Preprocessing. Before proceeding with Algorithm 1 to generate embedding vectors, we preprocess our data using the following steps. Firstly, we filter out the noise information by matching it with a set of regular expression rules (e.g., removing URLs, Stack Trace, and hex code). Then, we tokenize the textual information and

Chapter 3. CASMS: Combining Clustering with Attention Semantic Model for Identifying Security Bug Reports

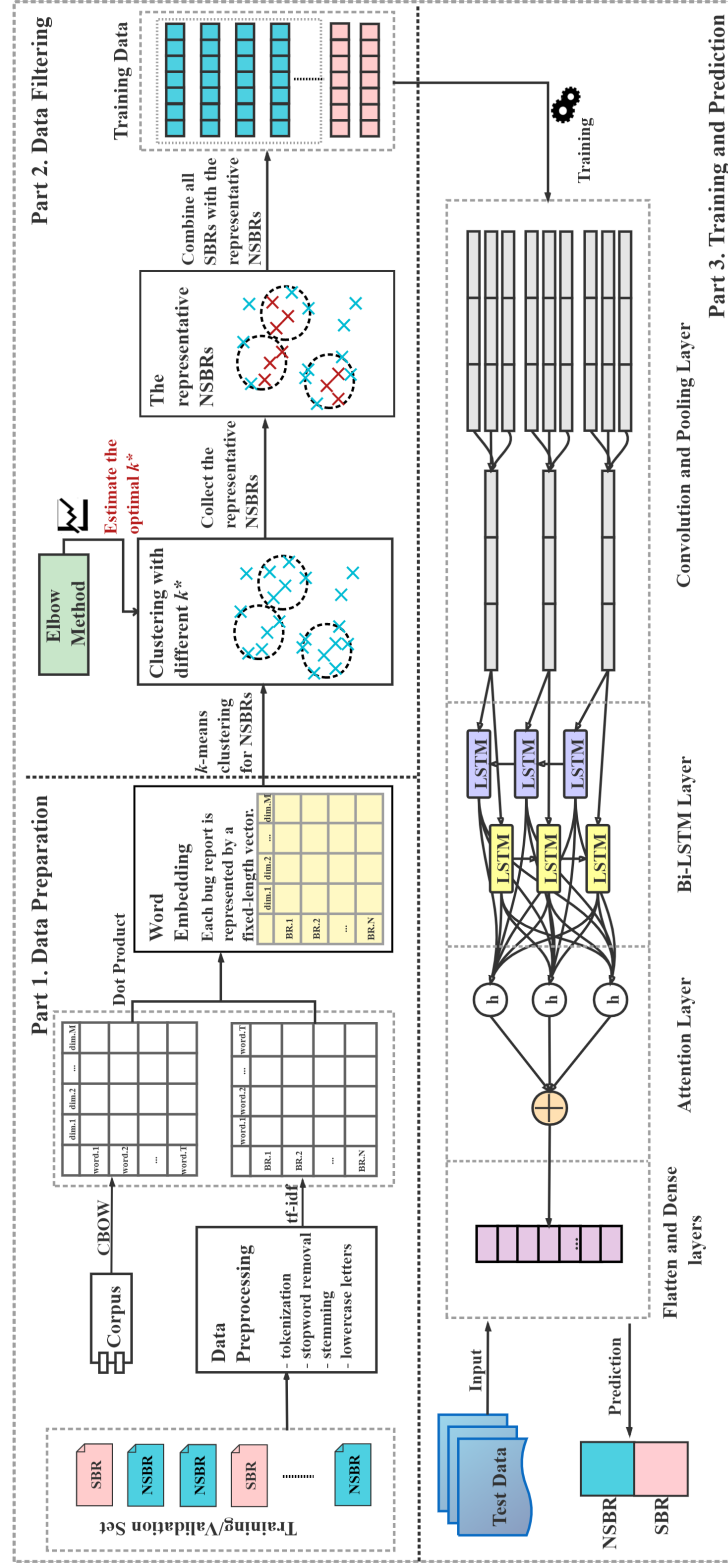


Fig. 3.2 The framework of CASMS.

remove the stop words. In addition, all terms are changed into lowercase, and they are stemmed.

Text representation. In Algorithm 1, CASMS calculates the $tf - idf$ weight of each term in the training set (Lines 1-4) and the vector of each term in the open-sourced corpus of collected bug reports [5] based on *word2vec* (Lines 5-8). Then, CASMS traverses all the terms in the training set (Line 9). If a term exists in the corpus (Line 10), the algorithm assigns the calculated word vector to this term (Line 11); otherwise, it will be represented by a zero vector of the same length (Line 12-13). Finally, through dot product by weights and vectors, each bug report in the generated matrix can be expressed as a same-length weighted vector (Line 16-17).

Algorithm 1 Calculate the vectors to represent bug reports.

CalVec(S, C, l)

Input: Training set S , bug report corpus C , the set length of each term l

Output: The vectors of bug reports M_{s2v}

```

1: for each term  $w$  in  $S$  do
2:    $M_{weight} \leftarrow Transform(tfidf(w))$ 
3:   // Generate the weight matrix of terms in  $S$ ;
4: end for
5: for each term  $t$  in  $C$  do
6:    $M_{w2c} \leftarrow Word2Vector(t)$ 
7:   // Return the matrix consisting of the vectors of each term in the corpus  $C$ ;
8: end for
9: for each term  $w$  in  $S$  do
10:  if  $w$  exists in  $C$  then
11:     $M'_{w2c}(w) \leftarrow M_{w2c}(w)$ 
12:  else
13:     $M'_{w2c}(w) \leftarrow M([0, ]*l)$ 
14:  end if
15: end for
16:  $M_{s2v} \leftarrow dot(M_{weight}, M'_{w2c})$ 
17: return  $M_{s2v}$ 
18: // Return the matrix of bug reports, each row is a vector to represent a bug report.

```

3.2.3 Data Filtering

Rough estimation on k values. As explained in Section 3.1.2, we choose the widely-used partition-based method k -means to cluster our data for filtering NSBRs.

To ensure the efficiency of our approach, we first estimate the range K of the number of clusters k and compute the recommended k^* using the Elbow method (Algorithm 2). For each chosen k , Algorithm 2 calculates the distance between each

Chapter 3. CASMS: Combining Clustering with Attention Semantic Model for Identifying Security Bug Reports

sample and its corresponding initialized cluster centroid. By summing and normalizing these distances, the average distortion for each k is determined. When the degree of distortion shows a significant improvement followed by a gentle improvement, the k value at this point is suggested for the subsequent clustering. To visualize the results, consider the example of the *Derby* project. A line graph illustrating k values and their corresponding distortions is generated (Lines 7-8), as shown in Figure 3.7. The turning point, marked with a dark blue arrow, is considered as k^* . For each project, we execute Algorithm 2 ten times, and the experimental results are summarized in Table 3.5.

Algorithm 2 Compute the recommended cluster numbers k for NSBRs.

CalClusterNum(K, S, C, l)

Input: The selected range K , training set S , corpus C , the set length of each term l

Output: The recommended k value

```

1:  $senVec \leftarrow CalVec(S, C, l)$ 
2: for  $k$  in  $K$  do
3:   classifier  $clf_k \leftarrow KMeans(k)$ 
4:    $clf_k \leftarrow fit(senVec_{nsbr})$ 
5:    $meanDistortion_k \leftarrow \sum_{n=1}^S (\frac{\min(cDist(senVec_{nsbr}, clf_{centroid}))}{S_{nsbr}})$ 
6: end for
7:  $line\ graph \leftarrow Draw(meanDistortion_k)$ 
8: // Draw a line graph based on the  $k$  and its mean distortion value;
9: return  $k^*$ 
10: // Return the  $k$  value at the turning point.

```

Clustering. The clustering conducted here is within a single class (NSBRs). As our datasets do not distinguish or label NSBRs, we utilize the clustering method within this single class to explore potential interrelations among features. Algorithm 3 is designed to find the suitable total number of selected NSBRs via automatic clustering. Given the range K , for each k in K , the algorithm initializes the centroid for each cluster (Line 3-4) and the zero matrices that include the index of the cluster to which each sample belongs and the distance from each sample to this cluster (Line 5-6). For each sample, the algorithm iteratively calculates and saves the index of its nearest cluster and the distance between it and the centroid of this cluster (Lines 11-21). During this process, if the saved index is the index of the nearest cluster for all samples, the centroids will no longer change (Line 10). Otherwise, the centroids will be changed gradually according to the average distance between the cluster centroid and all samples in this cluster (Lines 22-25). At last, each sample has its cluster index and the distance from the centroid. The NSBR set for succeeding training can be gathered according to different k values and different numbers n of samples in each cluster (Line 27). In this way, this

clustering algorithm finds diverse samples from each cluster. The number of clusters and the nearest NSBRs to each cluster centroid guarantee feature diversity.

Algorithm 3 Find diverse NSBRs from each cluster.

SelRepreNSBR(K, S, C, l, n)

Input: The selected range K , the training set S , corpus C , the set length of each term l , the number n of NSBRs in each cluster

Output: The chosen NSBRs for training

```

1:  $senVec \leftarrow CalVec(S, C, l)$ 
2: for  $k$  in  $K$  do
3:    $centroids_k \leftarrow ranSelect(k)$ 
4:   // Randomly chose the number of centroids  $k$  in  $S_{nsbr}$ ;
5:    $distMatrix[*][2]_k \leftarrow M_{init}$ 
6:   // each term includes the cluster's index and the distance from its centroid;
7:    $centroidChanged \leftarrow true$ 
8:   // set the initial state whether the centroids change;
9:   while  $centroidChanged == true$  do
10:     $centroidChanged \leftarrow false$ 
11:    for  $i \in S_{nsbr}$  do
12:      // Calculate the cluster index to which each NSBR  $i$  should belongs,
13:      // and the distance between the NSBR  $i$  and the cluster;
14:       $Index_i \leftarrow calIndex(senVec_{nsbr}, centroids_k)$ 
15:       $Dist_i \leftarrow calDist(senVec_{nsbr}, centroids_k)$ 
16:      if  $distMatrix_k(i, 0) \neq Index_i$  then
17:         $centroidChanged \leftarrow true$ 
18:      end if
19:       $distMatrix_k(i, 0) \leftarrow Index_i$ 
20:       $distMatrix_k(i, 1) \leftarrow Dist_i$ 
21:    end for
22:    if  $centroidChanged == true$  then
23:       $centroids_k \leftarrow updateCtd(disMatrix_k)$ 
24:      // Update the centroids in each cluster;
25:    end if
26:  end while
27:   $fltNSBR_k \leftarrow topN(centroids_k, disMatrix_k, n_k)$ 
28: end for
29: return  $fltNSBR_k$ 
30: // Return the retained NSBRs for different  $k$  and  $n$ .

```

3.2.4 Model Training and Prediction

Model Training. The CNN-BLSTM mechanism can excellently capture long-term dependencies, and the Attention mechanism can compensate for the former's neglect of different levels of attention to terms. In the clustering stage, each SBR is expressed as a $tf-idf$ -weighted word embedding vector for clustering and filtering. However, in the model training stage, to capture the sequential and semantic information, the

Chapter 3. CASMS: Combining Clustering with Attention Semantic Model for Identifying Security Bug Reports

relationship between terms in bug reports (SBR and NSBR) should be explored as we mentioned in Section 3.1.3. Thus, we consider each term in each retained text as a unit for further analysis. These terms are represented as word embeddings via the *word2vec*-based technique (the same as that is used in the clustering stage) to form our embedding layer, which requires that the vectors of terms should have the same length for further convolution calculation.

Due to the small size of our preprocessed datasets, in the front part of the whole model, we choose the shallow CNN-1D architecture to capture the contextual information to avoid high complexity or insufficient training of model parameters. Each term in the training set is calculated and padded into a d -dimensional sequence in Algorithm 1, and the variable-length sequence composed of d -dimensional vectors ($None, vector_dim$) is sent to the *embedding layer* as the input. The trained weights are also used to generate a 3D tensor ($None, vector_dim, embedding_size$), which is incorporated into the *convolution layer* with fixed-size kernels to capture features from temporal contextual information. This layer outputs a tensor with ($None, new_steps, filters$) shape. Followed by a *max pooling layer* with output shape ($None, downsampled_steps, filters$), the role of this pooling layer is to filter the features extracted by the model, which can not only remove some redundant information but also reduce the number of nodes in the network, thereby reducing the number of training parameters.

Afterward, a *Bidirectional LSTM layer* is added, taking the feature sequence calculated by the CNN architecture as input. In NLP, a significant feature of the text is serialization, and the order in which terms appear is often closely related to the semantics of the sentence. To make full use of the information of the sentence structure, we adopt a BLSTM model to extract the semantic information better, which applies memory cells to store the important information of long-range context. An LSTM unit illustrated in Figure 3.3 consists of an input gate i_t , a forget gate f_t , an output gate o_t , and a memory cell c_t that acts as an accumulator of the state information. This memory cell can be accessed, written, and cleared by those controlling gates. When a new input comes, if i_t is activated, the information will be accumulated to the cell. Similarly, if the f_t is turned on, the information of the last state c_{t-1} should be forgotten. The o_t controls whether the latest memory cell c_t will be propagated to the final state h_t . σ represents the logistic sigmoid activation function and W indicates the corresponding weight matrix from the cell to gate vectors. At each specific time node, the network can choose to remember or forget some information and send it to the next moment, and the steps are demonstrated by Equations 3.2-3.7 [187, 196].

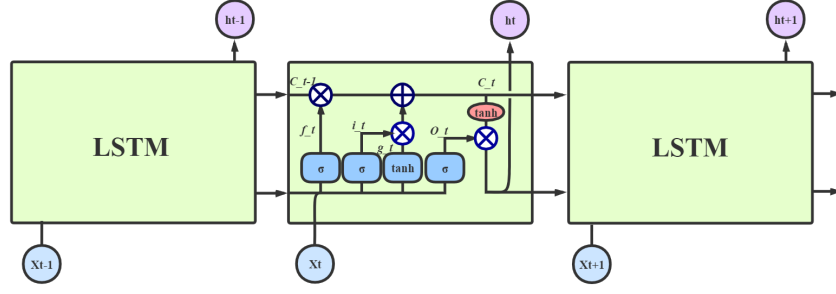


Fig. 3.3 The structure of the LSTM cell.

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \quad (3.2)$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \quad (3.3)$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \quad (3.4)$$

$$g_t = \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \quad (3.5)$$

$$c_t = f_t c_{t-1} + i_t g_t \quad (3.6)$$

$$h_t = o_t \tanh(c_t) \quad (3.7)$$

Since the network contains two sub-networks, which are forward and backward pass, respectively, the output of the i -th term is represented in Equation 3.8, and the shape of the output is $(None, timesteps, units \times 2)$.

$$h_i = \vec{h}_i \oplus \overleftarrow{h}_i \quad (3.8)$$

Because we adopt the self-attention model, the attention mechanism is able to dynamically generate the weights of different connections, which can handle variable-length information sequences. In our model, the input of our *Attention layer* can be expressed by three same fixed-length vectors query (Q), key (K), and value (V). By calculating the similarity score between two matrices (Q and K), applying the *softmax* function to activate it, and getting the weighted score of each input vector through dot product, we can finally gain the result keeping the same shape. This process is actually a manifestation of the Attention mechanism that eases the complexity of the prediction model. It is not necessary to input all input information into the next layer for calculation; only need to select some task-related information.

$$Attention(Q, K, V) = softmax(Q \cdot K^T) \cdot V \quad (3.9)$$

Prediction. In the end, the output of this stage is sequentially put into the *flatten* and *dense* layer with *sigmoid* function. The output is the predicted labels of the training data. Besides, the prediction model with the trained parameters is also saved for the next step. Through word embedding and padding, each term in the test data is represented by a fixed-length vector. These vectors are sent to the trained model and evaluated. The overall training and prediction processes can be found in Figure 3.2.

3.3 Experimental Setup

3.3.1 Research Questions

We employ three state-of-the-art methods, FARSEC, SMOTUEND, and LTRWES, as our baselines. All of them applied the commonly used machine learning algorithms (listed in Section 3.1.3) to build models.

Our experiments are performed by addressing the following research questions (RQs):

- **RQ1.** How effective is CASMS in classifying bug reports?
- **RQ2.** How does CASMS find the best results effectively and how robust is it based on the number of clusters and the total number of selected NSBRs?
- **RQ3.** Does selecting diverse samples through clustering improve the results?

For RQ1, we calculate the average of the top five best results for each project since these baselines extract the best results from multiple combinations between filters and classifiers. To make the comparison fair enough, we list the average of the top five best results of FARSEC, SMOTUEND, LTRWES, and CASMS. To answer the RQ2, we draw figures to illustrate 1) how the Elbow method finds the optimal number of clusters k^* and 2) the laws of the ratio of the two-class data corresponding to the best g -measure. To verify the effectiveness of retaining diverse data (answer the RQ3), we implement the idea of choosing NSBRs that are the most dissimilar to SBRs (held by our baselines) on the same k -means clustering algorithm.

3.3.2 Datasets

In the experiments, we use the same five datasets as the three baselines [2, 4, 5] for a fair comparison. The datasets were collected by Ohira et al. [197] with a Comma Separated

Table 3.1 Details of the collected datasets.

Project	Time Period	BRs	SBRs	SBRs(%)
Chromium	08/30/2008-06/11/2010	41,940	192	0.5
Camel	07/08/2007-09/18/2013	1,000	32	3.0
Ambari	09/26/2011-08/08/2014	1,000	29	3.0
Derby	09/28/2004-09/17/2014	1,000	88	9.0
Wicket	10/20/2006-11/09/2014	1,000	10	1.0

Value (CSV) format, where each row represents a bug report, and the columns describe the features of these reports including the metadata information and textual description (e.g., *issue id*, *priority*, *security*, *description*, and *summary*). Bug reports are classified according to the label of the security field, where 1 is for SBR and 0 is for NSBR. The details of datasets are shown in Table 3.1. We aggregate the information of description and summary for training, and an example is described in Figure 3.1. The text in the orange font is *summary*, and the text in the grey font is *description*. To calculate the $tf-idf$ vectors, for each project, the vocabulary is its training set. It aims to find the importance of words in the specific dataset. The large domain corpus used for implementing the word embedding technique in Algorithm 1 has been extracted from 1,490,066 bug reports and open-sourced in [5]. The five datasets and the corpus can be found in our GitHub¹.

3.3.3 Implementation

Our prediction model is implemented by using TensorFlow [198], an open-sourced deep learning library using Python. In the process of designing our model, parameter setting has a significant effect on its performance. We use binary *Cross-Entropy* for labeling the bug reports into two categories. In addition, different *optimizers* for iteratively updating the network weights and minimizing the loss are applied to enhance the performance, and *regularizers* are implemented to alleviate overfitting problems. To make the model more adaptive and general, the *model check point* technique is used to keep tracking the performance of the model on the validation dataset at each epoch.

The bug reports are equally divided into training/validation and test parts. For each project, 50% of the whole dataset is used for model training, and the remaining 50% is for model testing (prediction). During the model training stage, the K-fold cross-validation methodology is applied. Because our datasets are small, we choose K=3 for our experiments. The training/validation set is divided into three parts, one

¹<https://github.com/mkkmaomao/bug-report-prediction>

Chapter 3. CASMS: Combining Clustering with Attention Semantic Model for Identifying Security Bug Reports

Table 3.2 Hyperparameter tuning range for the prediction model.

Explanation	Architecture	Hyperparameter	Search Scope
Embedding	embed_size	[50, 100, 200]	The indexes representing terms are turned into vectors with the same length.
Convolution	filters	[64, 128]	The number of output filters in the convolution layer.
	kernel_size	[3, 5, 8]	A convolution kernel is used for producing a tensor of outputs; its size determines the length of the convolution window.
	activation	[tanh, relu, sigmoid, leak relu]	The activation function for the hidden layer.
B-LSTM	units	[32, 64, 128]	The dimension of the output of one LSTM layer (the BLSTM layer should have the units $\times$ 2-dimension).
	dropout	[0.1, 0.2, 0.5]	The fraction of the dropped units aiming to enhance the generalization ability.
	regularizer	kernel: $[l_1, l_2]$, bias: $[l_2]$	The regularizer functions applied to avoid overfitting.
Compiling	optimizer	[adam, rmsprop, adadelata, nadam]	The optimizers invoked for training.
	batch_size	[32, 64, 128]	The number of samples selected for one training, which affects the optimization degree and speed of the model.

part is kept as the data for validation, and the other two parts are used for training. The cross-validation is repeated three times, each validation is performed once, the results of three times are averaged, and a single estimate is finally obtained. The model with the highest average g -measure on the validation dataset is saved for prediction.

For hyperparameter tuning, to reduce the high computational cost of grid search by generating possible combinations among different hyperparameters, we first refer to the machine learning tool scikit-learn [199] in Python to estimate the configurations that are expected to contribute to better results. Then, we determine the search scope (shown in Table 3.2) and tune the hyperparameters successively.

Table 3.3 The evaluation metrics used in our study.

Metric	prediction	actual	
		SBRs	NSBRs
		TP	FP
	SBRs	FN	TN
	NSBRs		
pd	$TP/(TP+FN)$		
pf	$FP/(FP+TN)$		
g -measure	$(2 \times pd \times (100 - pf)) / (pd + (100 - pf))$		

3.3.4 Evaluation Metrics

We employ the three evaluation metrics that have been used in previous work [2, 4, 5]. Table 3.3 lists the definitions of the metrics to evaluate the performance of prediction models, where TP (True Positive) is the number of SBRs that are correctly predicted as SBRs, FP (False Positive) is the number of NSBRs that are wrongly predicted as SBRs, FN (False Negative) is the number of SBRs that are wrongly predicted as NSBRs, and TN (True Negative) is the number of NSBRs that are correctly predicted as NSBRs. The probability of detection (pd) is the ratio of the correctly predicted SBRs to the actual SBRs; The probability of the false alarm (pf) is the ratio of the NSBRs that are mislabeled as SBRs to all NSBRs; The g -measure [200] is the harmonic mean of pd and $(100-pf)$, where $100-pf$ represents the specificity (i.e., not predicting NSBRs as SBRs). The g -measure is a comprehensive metric, and more suitable for evaluating the prediction models trained on the unbalanced data [5]. Therefore, the g -measure is also considered as the optimization target for training our prediction model. To evaluate the robustness of our approach CASMS, we introduce the standard deviation (SD) indicator. In statistics, the standard deviation is a measure of the amount of variation or dispersion of a set of values [201]. A low standard deviation indicates that the values tend to be close to the mean (also called the expected value) of the set, while a high standard deviation indicates that the values are spread out over a wider range. The calculation of SD is shown below.

$$SD = \sqrt{\frac{1}{N} \sum_{i=1}^n (x_i - \mu)^2}, \quad (3.10)$$

where N is the number of experiments, x_i represents the value of g -measure in the i -th experiment, and μ indicates the corresponding average value of all experiments.

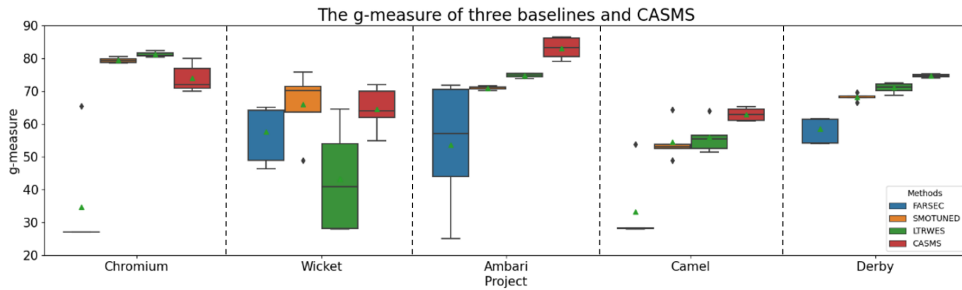


Fig. 3.4 The Boxplot of the g -measure values with CASMS and the three compared methods.

Chapter 3. CASMS: Combining Clustering with Attention Semantic Model for Identifying Security Bug Reports

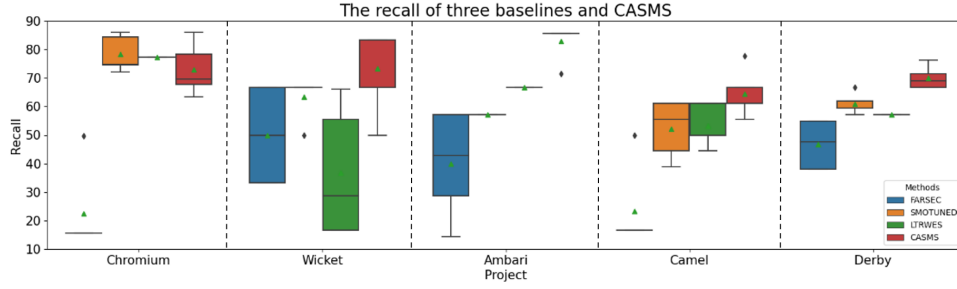


Fig. 3.5 The Boxplot of the pd values with CASMS and the three compared methods.

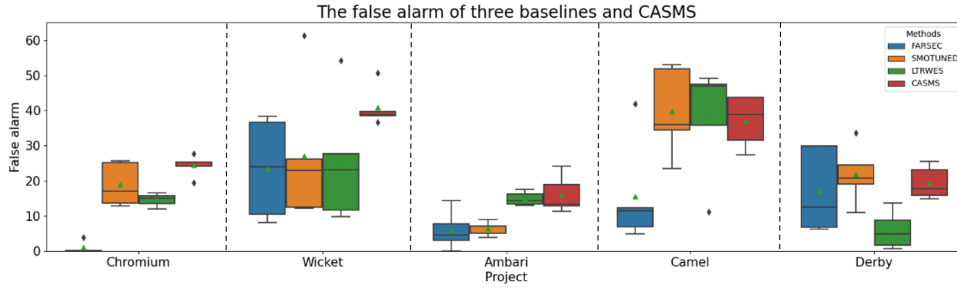


Fig. 3.6 The Boxplot of the pf values with CASMS and the three compared methods.

Table 3.4 Comparison of the average g -measure, recall, and false alarm of FARSEC, SMOTUNED, LTRWES, and CASMS.

Project	Metric	Approach			
		FARSEC	SMOTUNED	LTRWES	CASMS
Chromium	g -measure	34.79	79.38	81.20	74.00
	pd	22.48	78.42	77.40	73.05
	pf	0.96	18.84	14.59	24.41
Wicket	g -measure	57.69	66.02	43.18	64.60
	pd	50.00	63.36	36.74	73.33
	pf	23.49	26.98	25.29	40.85
Ambari	g -measure	53.73	70.92	74.79	83.12
	pd	40.00	57.1	66.70	82.85
	pf	5.93	6.41	14.87	16.10
Camel	g -measure	33.34	54.53	56.00	62.99
	pd	23.36	52.22	53.32	64.45
	pf	15.47	39.74	38.09	37.05
Derby	g -measure	58.59	68.15	71.02	74.72
	pd	46.68	60.94	57.10	70.00
	pf	17.07	21.74	5.93	19.44
Overall	g -measure	47.63	67.80	65.24	71.89
	pd	36.50	62.41	58.25	72.74
	pf	12.58	22.74	19.75	27.57

3.4 Results

3.4.1 RQ1. How effective is CASMS in classifying bug reports?

Methods: There are six filters (e.g., based on the Closest List Noise Identification (CLNI) algorithm) and five classifiers (i.e., Logistic Regression, Random Forest, Multilayer Perception, Naive Bayes, and K Nearest Neighbor) used in FARSEC and SMOTUNED, which generate the 30 ($=6 \times 5$) combination results. In each combination, there are different resampling ratios of NSBRs to SBRs based on their set threshold for each project. Similarly, LTRWES applies two kinds of selectors and six classifiers. For each project, in each of the 12 ($=2 \times 6$) combinations, the resampling ratio of NSBRs to SBRs is from 1:1 to 10:1. For these three approaches and CASMS, we illustrate the performance of the top five best results via Boxplots shown in Figure 3.4, 3.5, and 3.6, and we also detail the average of three metrics of the top five best results in Table 3.4.

Results: CASMS achieves the highest average g -measure value (71.89%) on the five projects, and outperforms FARSEC, SMOTUNED, and LTRWES by 24.26%, 4.09%, and 6.65%, respectively.

As shown in Figure 3.4, SMOTUNED and LTRWES perform better than CASMS in g -measure and far surpass FARSEC on the *Chromium* project. Compared with FARSEC and LTRWES, SMOTUNED and CASMS perform well on the *Wicket* project with a smaller variance. On the remaining projects *Ambari*, *Camel*, and *Derby*, CASMS has outstanding performance, which outperforms three baselines in both the mean and median values of g -measure. Since the primary purpose of SBR detection is to predict the actual SBRs more correctly, SBRs usually have the priority to be detected than NSBRs. Therefore, the pd is a more useful metric than pf , as it describes the ability of the prediction model to identify the actual SBRs from all bug reports. Figure 3.5 displays the recall values (pd) of three methods and CASMS. Except for the *Chromium* project where SMOTUNED performs slightly better than CASMS, on the remaining four projects, CASMS significantly outperforms other methods with an ideal small variance. Similarly, Figure 3.6 illustrates the performance of three baselines and our approach in false alarm values (pf). On the *Chromium* and *Wicket* projects, CASMS has a relatively high average pf compared with other methods, but on the *Ambari*, *Camel*, and *Derby* projects, CASMS has a similar or even lower pf . It implies that CASMS achieves good performance on the g -measure and pd evaluation metrics at the expense of the performance of the pf metric to some extent, and the relatively

Chapter 3. CASMS: Combining Clustering with Attention Semantic Model for Identifying Security Bug Reports

high pf is still an issue in identifying security bug reports, which needs to be improved in our future work.

Intuitively, Table 3.4 lists the average g -measure, pd , and pf values of the five best combination results of FARSEC, SMOTUNED, LTRWES, and CASMS. Although the compared methods outperform CASMS (27.57%) by 4.83%-14.99% in terms of pf , CASMS is able to identify SBRs more accurately with high pd (72.74% on average), which outperforms FARSEC, SMOTUNED, and LTRWES by 36.24%, 10.33%, and 14.49%, respectively. In terms of g -measure, CASMS performs better than the three baselines on the three projects *Ambari*, *Camel*, and *Derby* by 8.33%-29.39%, 6.99%-29.65%, and 4.09%-16.13%, respectively. On the *Wicket* project, the g -measure value of CASMS is 6.91% and 21.42% higher than that of FARSEC and LTRWES, separately, and only 1.42% lower than that of SMOTUNED. On the *Chromium* project, the performance of CASMS is 39.21% better than FARSEC, but worse than SMOTUNED and LTRWES by 5.38% and 7.2%, respectively.

Answer to RQ1: CASMS is superior to FARSEC, SMOTUNED, and LTRWES in average g -measure and recall.

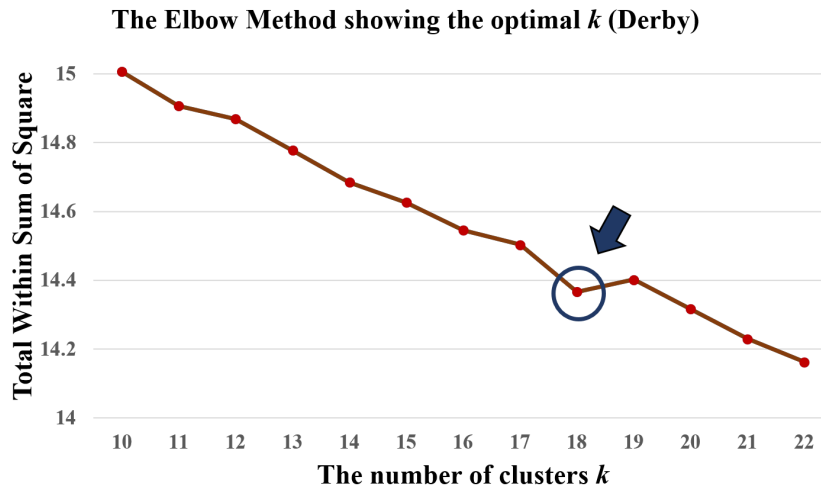


Fig. 3.7 Selection of an optimal k^* in the *Derby* project.

Table 3.5 Comparison of the recommended k^* values and the experimental k values yielding the best g -measure.

Project	Recommended k^*	Calculated k with best performance
Chromium	18, 20	15, 18
Wicket	4	4 , 5
Ambari	4, 8	4, 8
Camel	4	3, 4
Derby	18, 19	18, 19

3.4.2 RQ2. How does CASMS find the best results effectively and how robust is it based on the number of clusters and the total number of selected NSBRs?

Methods (1): To avoid repeated experiments, CASMS roughly estimates the optimal number of clusters k^* using the Elbow algorithm (shown in Algorithm 2). To show the validity of the coarse estimate, the recommended k^* values are calculated and listed in the 2nd column of Table 3.5. The 3rd column lists the k values when the g -measure reaches the optimal value, which is obtained through multiple experiments. Figure 3.7 takes *Derby* dataset as an example, showing one of the recommended k^* that is equal to 18 (is circled and marked with a dark blue arrow).

Results (1): In Table 3.5, the k^* calculated ten times is 4 or 8 for the datasets with a few SBRs (*Camel*, *Ambari*, *Wicket*), and for other datasets with more SBRs (*Chromium* and *Derby*), the range of k^* is between 18 and 20.

As shown in the 3rd column of Table 3.5, it can be seen that the model achieves the best performance within a certain range of k value for each dataset. The three projects (*Camel*, *Ambari*, *Wicket*) containing fewer SBRs for training have k values ranging from 3 to 8, and the other two projects (*Derby* and *Chromium*) with more SBRs have k values ranging from 15 to 19. We can find that when the prediction results for each project are optimal, the values of k almost completely cover the value of k^* we estimated in advance (marked in bold). For example, in the best cases, the k values of *Ambari* (4 and 8) and *Derby* (18 and 19) totally coincide with the pre-calculated k^* .

Methods (2): To explore whether there is a certain rule for choosing the resampling ratio of two classes and whether it is robust, we calculate the best g -measure of different ratios and the corresponding indicator for evaluating the stability of CASMS. Table 3.6 shows the number of NSBRs and SBRs in the training set when the two best results reach, and the ratio of these two classes. Figure 3.8 displays how the g -measure changes as the percentage of retained training NSBRs increases for each project while

Chapter 3. CASMS: Combining Clustering with Attention Semantic Model for Identifying Security Bug Reports

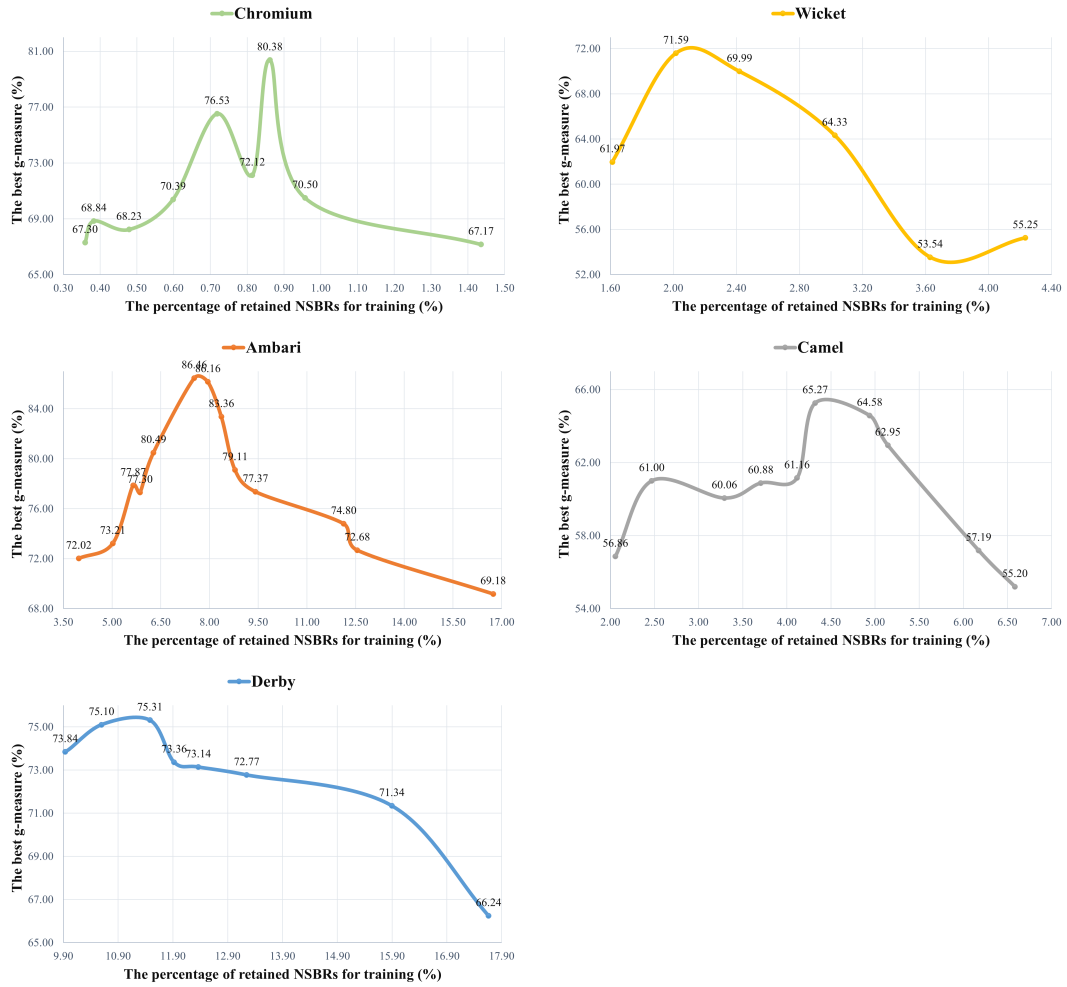


Fig. 3.8 Changes in the best g -measure with the number of NSBRs retained.

Table 3.6 The NSBR to SBR ratio leading to the best g -measure.

Project	NSBRs	SBRs	Ratio
Chromium	180	77	2:1-3:1
	150	77	1:1-2:1
Wicket	10	4	2:1-3:1
	12	4	3:1
Ambari	36	22	1:1-2:1
	38	22	1:1-2:1
Camel	21	14	1:1-2:1
	24	14	1:1-2:1
Derby	52	47	1:1-2:1
	47	47	1:1

the range of k is from 2 to 20. Furthermore, we compare the three indicators (average, standard deviation, and median) between LTRWES and CASMS, and the standard deviation of g -measure is used to evaluate the robustness of these two approaches.

Results (2): The relationship between the percentage of retained NSBRs and the g -measure metric is illustrated in Figure 3.8. It is evident that the four lines of *Wicket*, *Camel*, *Ambari*, and *Derby* have a general trend that it increases rapidly before reaching the highest g -measure value, followed by a monotonic decline. Table 3.6 lists the number of selected NSBRs where the peak values of g -measure reach, and the corresponding percentage of retained NSBRs are 2.02%, 4.32%, 7.53%, and 11.48%, respectively. Since the remaining project *Chromium* has more SBRs for training, this may make the choice of the number of NSBRs fall into a locally optimal solution. There is not much difference between the local optimal g -measure and the globally optimal g -measure, with a difference of 3.85%. The percentage corresponding to the optimal number of NSBRs is 0.86%.

Overall, in all the optimal cases, the ratio of NSBRs to SBRs for training is between 1:1 and 3:1. *Chromium* has the ratio range from 1:1 to 3:1, *Wicket* has the ratio range from 2:1 to 3:1, and the others have the ratio range from 1:1 to 2:1.

Table 3.7 Comparison of the average avg (%), standard deviation sd (%), and median md (%) of g -measure of LTRWES and CASMS.

Project	Indicator	LTRWES		CASMS
		rs -filter	ms -filter	
Chromium	avg	79.23	53.20	71.22
	sd	2.17	16.95	4.52
	md	79.17	58.83	70.00
Wicket	avg	13.25	32.20	62.83
	sd	11.74	15.49	7.44
	md	8.35	27.91	62.42
Ambari	avg	69.28	67.05	77.69
	sd	3.50	9.56	5.39
	md	69.60	69.38	77.37
Camel	avg	47.57	44.69	60.51
	sd	7.19	11.49	3.31
	md	47.00	48.25	60.70
Derby	avg	61.20	66.16	72.99
	sd	5.78	6.58	2.41
	md	61.00	67.40	73.36
Overall	avg	54.11	52.66	69.05
	sd	6.08	12.01	4.61
	md	53.02	54.35	68.77

Method (3): In order to evaluate the robustness of CASMS, for each project, we calculate the standard deviation of g -measure in LTRWES and CASMS. The

Chapter 3. CASMS: Combining Clustering with Attention Semantic Model for Identifying Security Bug Reports

average and median indicators are also introduced for comparison. Since LTRWES conducted experiments under the ratio of two classes from 1:1 to 10:1, we calculate the g -measure under the ratio span and choose the best-performing SVM as the classifier in LTRWES. For CASMS, we use the results under the ratio span between 1:1 and 3:1 for comparison. In table 3.7, to avoid too many decimals, the ”%” of the three indicators (average, standard deviation, and median) shown here is omitted.

Result (3): The best results are marked in bold, and we have two main findings. The first finding is that the maximum standard deviation of CASMS is 7.44%, and that of LTRWES is 16.95%. For the three projects (*Wicket*, *Camel*, and *Derby*), the standard deviation of CASMS is much lower than that of LTRWES. For the other projects (*Chromium* and *Ambari*), when LTRWES uses rs -filter, the standard deviation of CASMS is 2.1 and 1.5 times that of LTRWES, respectively; When ms -filter is used, the standard deviation of LTRWES is 3.8 and 1.8 times that of CASMS, respectively.

The second finding is that the average and the median of CASMS on the four projects (except *Chromium*) are higher than LTRWES, and the gap between CASMS and LTRWES is large. On the *Wicket* project, the median and the average g -measure of CASMS exceed those of LTRWES by 49.58% and 54.07%, respectively. On the *Chromium* project, CASMS does not perform as well as LTRWES, but the difference in performance between CASMS and LTRWES is small, i.e., the median and average of CASMS are 8.01% and 9.17% lower than those of LTRWES.

On the whole, the standard deviation of g -measure of CASMS is 1.47% and 7.4% lower than that of LTRWES with rs -filter and ms -filter, respectively. The average and median of g -measure of CASMS are higher than those of LTRWES by 14.94%-16.39% and 14.42%-16.75%.

Answer to RQ2: CASMS can narrow the range of cluster numbers based on the Elbow method and quickly find the optimal number of NSBRs (should be one to three times the number of SBRs for projects), which indicates the effectiveness. In addition, the robustness of CASMS is better than that of LTRWES.

3.4.3 RQ3. Does selecting diverse samples through clustering improve the results?

Methods: To be more rigorous, we also test the idea of choosing the NSBRs with the most dissimilar to SBRs held by our baselines through the k -means clustering

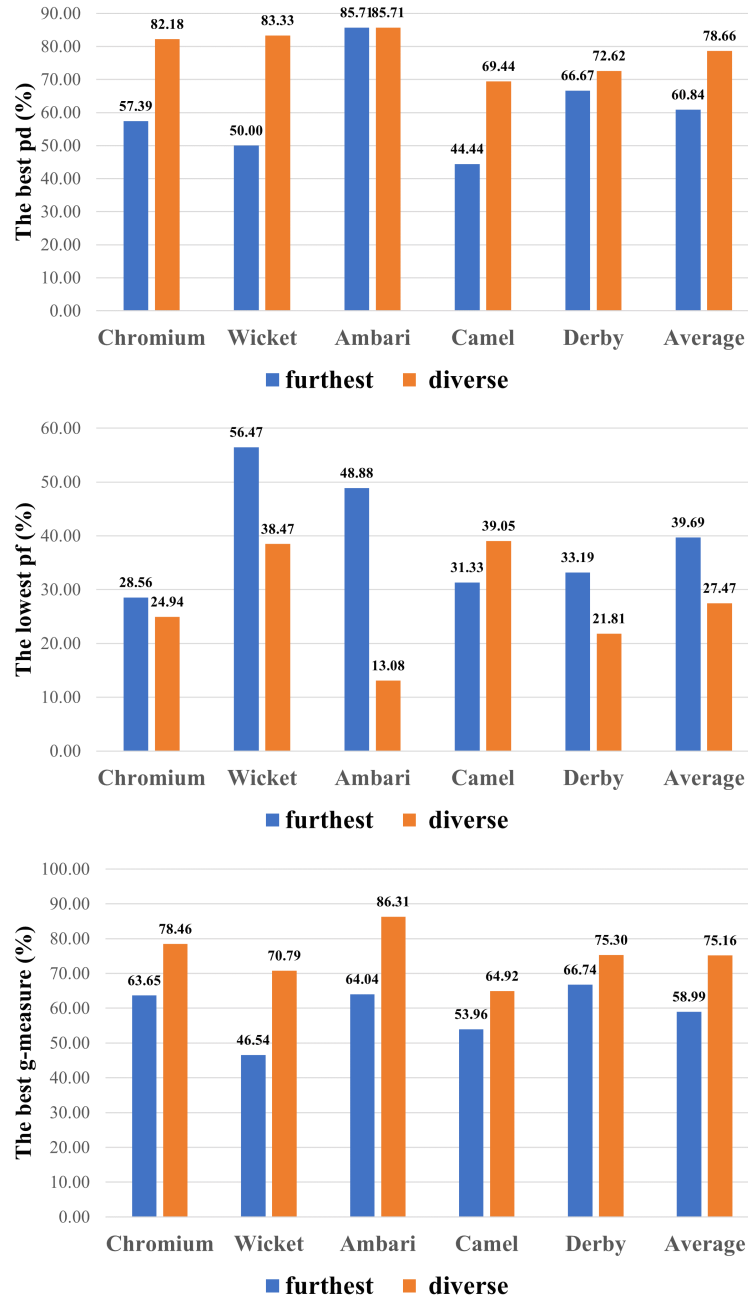


Fig. 3.9 The best pd , pf , and g -measure based on two data selection strategies.

algorithm. Both NSBRs and SBRs are used for clustering. When the value range of k is 2 to 20, there are 19 groups of NSBRs. For each value of k , we take the union of the NSBRs in the selected clusters that contain fewer SBRs but more NSBRs. For example, if the ratio of NSBRs to SBRs in a cluster is greater than 10, the NSBRs in the cluster will be gathered. In other words, if there are many NSBRs but a few SBRs in a cluster, these NSBRs in this cluster will be regarded as the most dissimilar to SBRs. In order to keep the ratio of NSBRs to SBRs between 1:1-3:1, we then take the intersection of the NSBRs obtained when k is a different value. After that, the intersecting NSBRs in these clusters are integrated with all SBRs for training.

In detail, after each clustering with different k , two dictionaries (pairs of *key* and *value*) are generated for NSBRs and SBRs. In each dictionary, the pairs are arranged in reverse order according to the number of samples in each cluster. *key* refers to the cluster index assigned to the samples, and the *value* is the ranking of the number of samples corresponding to the cluster. The algorithm then traverses the same cluster index in each pair of two dictionaries to see if the cluster meets a certain condition. For example, if the difference between the ranking of the same cluster in the dictionary of SBRs and in that of NSBRs is greater than a certain value, the cluster indexes will be stored. Finally, the NSBRs in the selected clusters with different k values are intersected, and the final filtered NSBR data is obtained. If the number of screened samples is insufficient, more samples can be filtered out by adjusting the threshold. Finally, for each dataset, the number of selected NSBRs that are the most dissimilar to SBRs and the number of diverse NSBRs are kept in the same range, e.g., the ratio of selected NSBRs to SBRs is between 1:1 and 3:1.

In Figure 3.9, the best g -measure values based on two data selection strategies are illustrated by a bar graph. The orange bars represent the results of retaining the diverse NSBRs, and the blue bars indicate the results of retaining the NSBRs that are the furthest to SBRs.

Results: When adopting the prediction model with the same parameters (Section 3.2.3), the best results in terms of pd , pf , and g -measure for each project are not as good as the method of retaining diverse NSBRs. In Figure 3.9, the best pd , pf , and g -measure of the two data selection strategies are compared, and the strategy of retaining the diverse NSBRs performs better in each of these metrics. Based on the strategy of retaining the dissimilar NSBRs, the overall g -measure and pd values are nearly 16.2% and 17.8% lower than those of CASMS, and the pf is nearly 12.2% higher than that of CASMS. This indicates that owing to the high feature overlapping,

even if the NSBRs in the clusters that are farther from most of the SBRs are retained, it is still challenging to achieve the desired effect we expect.

Answer to RQ3: The performance of extracting the furthest NSBRs from SBRs is not a patch on that of selecting diverse NSBRs.

3.5 Threats to Validity

3.5.1 Construct Threats

One of the threats is the choice of evaluation metrics, which is very important for statistical analysis. Since detecting SBRs has priority in bug report classification, the g -measure and recall are designed to be the widely used evaluation metrics. However, there are different numbers of combinations between filters and classifiers for our baselines (30 combinations of FARSEC and SMOTUNED and 12 combinations of LTRWES). If we only choose the best result for comparison, it is not reasonable, e.g., compare a model that has spent a lot of effort on multiple parameter optimizations with a model with default parameters. To counter this threat, we calculate the average of the best five experiment results for comparison. The Boxplots are drawn to comprehensively compare the performance between methods, which includes a number of indicators, such as the four quantiles and outliers on different metrics. In addition, data scarcity is another construct threat, such as the lack of cybersecurity-related descriptions in the bug reports. In the four projects, the dataset is relatively small, and even some of the samples have a short description related to cybersecurity. To the best of our knowledge, there is no related work to effectively solve this threat. Therefore, it is considered to be included in our further work.

3.5.2 Internal Threats

The first threat comes from the uncertainty of clustering. The centroids of clusters may be changed due to the random initialization setting. To eliminate the possible bias, we execute the clustering algorithm five times for each project and extract different numbers of NSBRs. The selected samples for each time have a high repetitive rate of more than 80%, and the predicted results based on these samples have shown similar trends. Thus, the centroids formed by each clustering will not have a visible impact on the selected training data and the prediction results. The second one is that the

Chapter 3. CASMS: Combining Clustering with Attention Semantic Model for Identifying Security Bug Reports

performance of CASMS is sensitive to the selection of the cluster and NSBR numbers. To mitigate the threat, we conduct numerous experiments with a comprehensive combination of two quantities (Figure 3.8) and calculate the recommended cluster numbers ten times for each project (Table 3.5) to ensure that the results shown in Table 3.4 are scientific and well-tried.

3.5.3 Conclusion Threats

Normally in experiments in similar fields, the ratio of training to test sets is set to 7:3 or 8:2 because abundant data is needed for training. In our datasets, there are limited samples (i.e., there are only a thousand samples in each of the four projects), especially there are a few SBRs in each project. Thus, not only do we need a certain number of SBRs for training, but we also need enough SBRs in the test set to evaluate the prediction results. We have tried creating more SBR samples to alleviate this threat via synthesizing new SBR samples or copying existing SBRs, but the results were not ideal and required continuous tuning. Finally, we divide the ratio of training and test sets into 1:1, which is the same as our three baselines.

3.5.4 External Threats

The external threat relates to the generalizability of the findings to other datasets. The facts that (1) the dataset used in this study is constructed by five projects containing 45,940 bug reports and (2) the projects were collected over multiple years, allow for a degree of external validation. Despite the dataset reflecting the class imbalance problem in bug reports and similar conclusions being drawn across all five projects, the model generated may not be applicable in a more complex scenario. Hence, this work shows that the proposed technique is valid and outperforms other approaches under certain conditions, and its generalization for different datasets will be subject to scrutinization in future studies.

3.6 Chapter Summary

In this chapter, we have introduced CASMS, a novel approach designed for detecting SBRs. Addressing the key challenges in bug report classification, including high feature overlap among data with different labels and the inability to capture contextual and sequential information, CASMS employs a k -means clustering algorithm in

conjunction with the Elbow algorithm. This combination enables the collection of the closest NSBRs to cluster centroids, serving as diverse samples for training. The successful utilization of the Elbow algorithm facilitates the identification of the optimal k for clustering. Furthermore, CASMS integrates a CNN-BLSTM attention model to extract semantic and sequential information, thereby enhancing prediction outcomes. Experimental results demonstrate that CASMS outperforms baselines by 4.09% to 24.26% and 10.33% to 36.24% in terms of average g -measure and recall, respectively. Notably, CASMS achieves these results with a limited number of experiments while maintaining superior robustness.

Chapter 4

AttSum: A Deep Attention-Based Summarization Model for Bug Report Title Generation

Concise and precise bug report titles help software developers to capture the highlights of the bug report quickly. Unfortunately, it is common that bug reporters do not create high-quality bug report titles. Recent LSTM-based Seq2Seq models were proposed to generate bug report titles automatically, but the text representation method and LSTM employed in such models make it difficult to capture the accurate semantic information and draw the global dependencies among tokens effectively. In this chapter, we propose a deep attention-based summarization model (i.e., AttSum) to generate high-quality bug report titles. Specifically, the AttSum model employs the encoder-decoder framework, which utilizes the robustly optimized bidirectional-encoder-representations-from-transformers approach to encode the bug report bodies to capture contextual semantic information better, the stacked transformer decoder to automatically generate titles, and the copy mechanism to handle the rare token problem.

To validate the effectiveness of AttSum, we conduct automatic and manual evaluations on 333563 “<body, title>” pairs of bug reports and perform a practical analysis of its ability to improve low-quality titles. The result shows that AttSum is superior to the state-of-the-art baselines by a substantial margin both on automatic evaluation metrics (e.g., by 3.4%–58.8% and 7.7%–42.3% in terms of recall-oriented understudy for gisting evaluation in F1 and bilingual evaluation understudy, separately) and three human-set modalities (e.g., by 1.9%–57.5%). Moreover, we analyze the

impact of the training data size on AttSum and the results imply that our approach is robust enough to generate much better titles. This work was published in IEEE Transactions on Reliability in 2023 [202].

The subsequent sections of this chapter are organized as follows: Section 4.1 introduces the methodology of the proposed approach AttSum. Section 4.2 describes the experimental setup, and Section 4.3 demonstrates the results and analysis. Section 4.4 discusses the automated evaluation metrics, AttSum’s effectiveness, and insights for further research. Section 4.5 presents the threats to the validity. Finally, Section 4.6 summarizes the work of this chapter.

4.1 Methodology

4.1.1 Framework

The workflow of our approach is demonstrated in Figure 4.1. The model architecture is composed of two main components: a pre-trained **RoBERTa Encoder** and a **stacked Transformer Decoder**. To handle the rare token problem of the title generation task, we incorporate a *copy* mechanism to facilitate copying some necessary tokens from the descriptive body to the target generated title. (In this paper, we treat “tokens” and “terms” as interchangeable.) The organized “<body, title>” pairs are preprocessed as sequences with special tokens. In order to be consistent with the data format in our pre-training model, we add “*s*” and “/*s*” and “*SOS*” and “*EOS*” to each body sequence $\mathbf{x}$ and title sequence $\mathbf{y}$, separately, and these two pairs of special tokens perform similar effects. Hence, the token sequences of body and title can be represented as $\mathbf{x} = [< s >, x_1, x_2, \dots, x_m, \dots, < /s >]$ and $\mathbf{y} = [< SOS >, y_1, y_2, \dots, y_k, \dots, < EOS >]$, respectively. In addition, the input body $\mathbf{x}$ and the targeted title $\mathbf{y}$ share the same word embedding based on the pre-trained RoBERTa tokenizer, and then feed them into our RoBERTa encoder and stacked vanilla Transformer decoder respectively for training. When our approach achieves the best performance on the validation set, it is saved for generating bug report titles on the test set.

4.1.2 RoBERTa Encoder

BERT (Bidirectional Encoder Representations from Transformers) [83] has excellent performance on multiple tasks. It is a state-of-the-art masked language model that

Chapter 4. AttSum: A Deep Attention-Based Summarization Model for Bug Report Title Generation

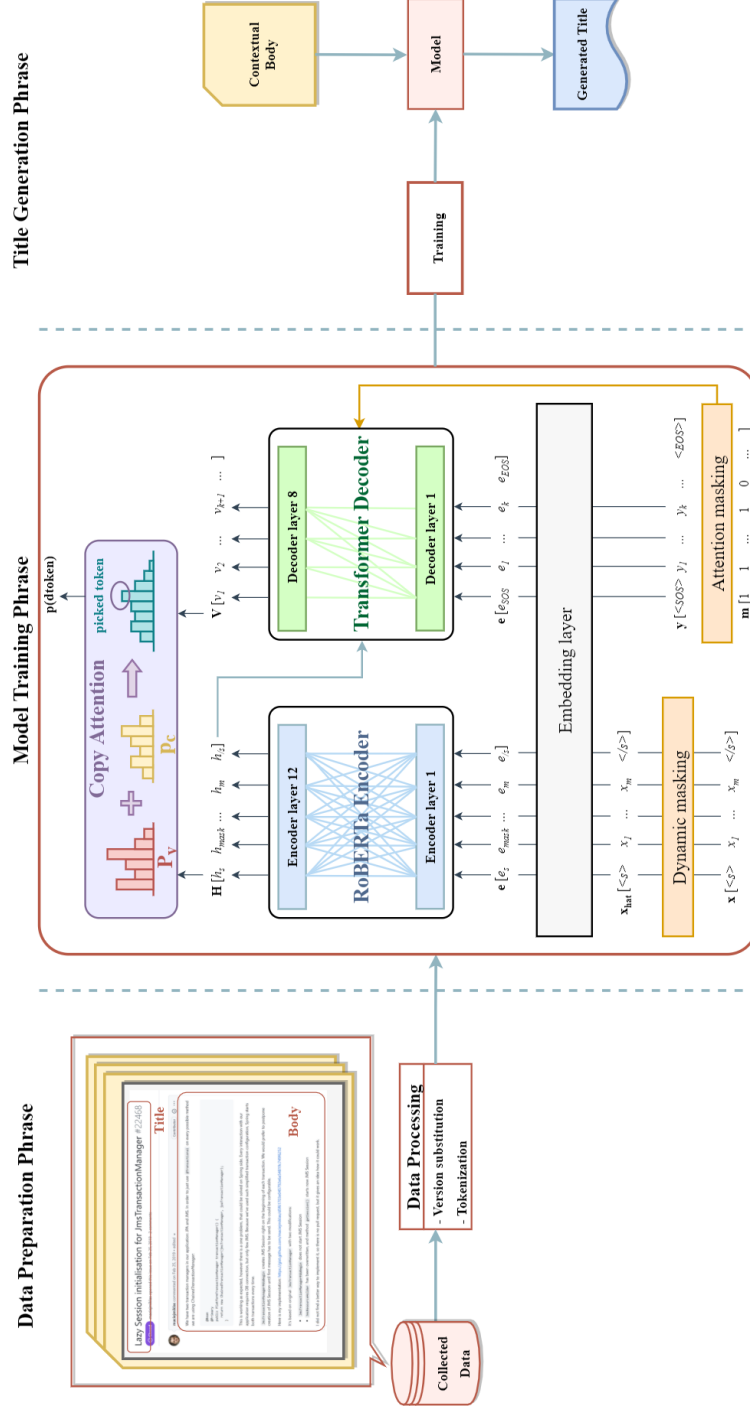


Fig. 4.1 The three-phase framework of our approach.

adopts transformer encoder architecture with multiple attention heads to capture long-distance dependencies and semantic relationships. Nevertheless, some recent research works [203, 204] have doubts about the necessity of adopting NSP (Next Sentence Prediction) used by BERT since the experimental results could not show consistent improvement in their studies. Therefore, following RoBERTa [95] and other latest works (e.g., [204]), we remove the NSP objective and make each segment contain complete sentences contiguously sampled from one or more documents.

The RoBERTa encoder performs pre-training by maximizing the likelihood of the conditional probability $p_\theta(\bar{x}|\hat{x})$, i.e., to recover the original tokens $\bar{x}$ from the corrupted text sequence $\hat{x}$. The pre-training objective can be expressed as follows[203]:

$$\begin{aligned} \max_{\theta} \log p_\theta(\bar{x}|\hat{x}) &\approx \sum_{t=1}^T m_t \log p_\theta(x_t|\hat{x}) \\ &= \sum_{t=1}^T m_t \log \frac{\exp(H_\theta(\hat{x})_t^\top e(x_t))}{\sum_{x'} \exp(H_\theta(\hat{x})_t^\top e(x'))}. \end{aligned} \quad (4.1)$$

We suppose the length of an input text sequence x is T . $\hat{x}$ denotes the dynamical masking version of the input sequence with a special symbol $[MASK]$. The masked tokens are expressed as $\bar{x}$, and x_t is masked where $m_t = 1$. $e(x)$ indicates the word embedding of x , which is fed into the encoder H_θ and gets the $H_\theta(x)$ that refers to the sequence of hidden vectors h_i . During the fine-tuning period, the pre-trained RoBERTa is employed as our encoder, and the parameter θ in Equation (4.1) is updated to fit our training data, i.e., as we mentioned, a body sequence and its corresponding title can be expressed as $\mathbf{x}$ and $\mathbf{y}$, respectively.

$$\begin{aligned} H_\theta(\mathbf{x}) &= [H_\theta(\mathbf{x})_1, H_\theta(\mathbf{x})_2, \dots, H_\theta(\mathbf{x})_T] \\ &= H_\theta[h_1, h_2, \dots, h_T] \\ &= ENCODER(\mathbf{x}), \end{aligned} \quad (4.2)$$

where H is a Transformer encoder that is able to capture bidirectional contextual information via implementing bidirectional self-attention operations. Our RoBERTa encoder is composed of a 12-layer bidirectional Transformer encoder. The output of the last layer is denoted as $H_\theta(\mathbf{x})$, which is then passed to our decoder and the copy mechanism for further processing.

4.1.3 Transformers Decoder

We stack L -layer vanilla Transformer decoders [31] with the multi-head self-attention mechanism to construct our decoder. We gather together the output $H_\theta(\mathbf{x})$ of our encoder and the token sequence $\mathbf{y}$ for the next token generation. In Figure 4.1, we suppose the decoding step at this time is in the state of generating the $(k+1)th$ token.

To avoid involving the tokens that are after the $(k+1)th$ token in the self-attention operation during training, a special mask expressed as $m = [\mathbf{1}, \mathbf{1}, \dots, \mathbf{1}, 0, \dots]$ is used for masking, where the positions of the front $(k+1)$ are “1”s and that of the following are “0”. In this way, the previous k tokens that have been generated are used to predict the next $(k+1)th$ token. All tokens can be generated simultaneously in accordance with the masked self-attention.

Consequently, we collect the output $H_\theta(\mathbf{x})$ of the encoder, the masked sequence m , and the embedding vectors $E(\mathbf{y}) = E[e_{EOS}, e_1, e_2, \dots, e_k, \dots, e_{EOS}]$ that are derived from the input sequence $\mathbf{y}$, to feed into our decoder. The obtained matrix $V_{\theta'}(H_\theta(\mathbf{x}), E(\mathbf{y}), m)$ is a sequence of the hidden vectors of the predicted tokens.

$$\begin{aligned} V_{\theta'}(H_\theta(\mathbf{x}), E(\mathbf{y}), m) &= V_{\theta'}[v_1, v_2, \dots, v_{k+1}] \\ &= DECODER(H_\theta(\mathbf{x}), E(\mathbf{y}), m), \end{aligned} \quad (4.3)$$

where the vector v_{k+1} refers to the representation of the $(k+1)th$ predicted token generated by the decoder. After that, the embedding vector e_k and the generated vector v_{k+1} are then passed to the following copy attention layer.

4.1.4 Copy Mechanism

The motivation for adopting the copy mechanism is that some tokens are rarely seen in a corpus but are important for a specific bug report [205–207]. According to our statistics, some tokens only appear dozens of times or even a few times, such as application frameworks, programs, variables and functions, and so on. In such cases, it is hard for a decoder to generate such rare items compared to high-frequency items that occur tens of thousands of times. Thus, we incorporate the copy mechanism to handle the rare token occurrence for bug report title generation. Following the architecture of the pointer-generator networks proposed by See et al.[208], we first calculate the Encoder-Decoder attention score a_{k+1} . On the basis of it, we then figure out the copy probability p_c as a binary classifier to determine whether to copy the token directly from

the input sequence or not. Suppose we still prepare to generate the $(k+1)th$ token, the encoder hidden states $H_\theta(\mathbf{x})$, the embedding vector e_k , and the decoder hidden state v_{k+1} are fed into the copy mechanism together as its inputs:

$$d_i^{k+1} = v^T \tanh(W_h h_i + W_v v_{k+1} + b_{att}), \quad (4.4)$$

$$a_i^{k+1} = \text{softmax}(d_i^{k+1}), \quad (4.5)$$

$$c_{k+1}^* = \sum_{i=1, \dots, |x|} a_i^{k+1} h_i, \quad (4.6)$$

where v^T , W_h , W_v , and b_{att} are model parameters that need to be learned. Equation (4.4) calculates the attention score d_i^{k+1} between the current decoder hidden state v_{k+1} and the encoder hidden state h_i of the i -th token using the Tanh activation function. This helps to keep the score within a bounded range (-1 to 1), which stabilizes the values passed to the Softmax layer. Equation (4.5) applies the Softmax function to the attention scores d_i^{k+1} to generate a probability distribution a_i^{k+1} . This distribution reflects how much attention each token in the input sequence should receive relative to the current decoding step. Equation (4.6) uses the attention weights a_i^{k+1} to compute the context vector c_{k+1}^* . This context vector is a weighted sum of the encoder hidden states, emphasizing those that are more relevant for predicting the next token. The context vector c_{k+1}^* is then used to derive the probability distribution P_v (Equation (4.7)) over the vocabulary tokens and the copy probability p_c (Equation (4.8)) for tokens directly from the input sequence $\mathbf{x}$. The combined final probability distribution $p^*(dtoken)$ (Equation (4.9)) for decoding the $(k+1)$ -th token is calculated, leveraging both the vocabulary and the original input sequence for a more robust output prediction. The calculation is shown below:

$$P_v = \text{softmax}(W_v[v_{k+1}, c_{k+1}^*] + b_v), \quad (4.7)$$

$$p_c = \text{sigmoid}(w_{c^*}^T c_{k+1}^* + w_v^T v_{k+1} + w_e^T e_k + b_c), \quad (4.8)$$

$$p^*(dtoken) = p_c \sum_{i:w_i=dtoken} a_i^{k+1} + (1 - p_c)P_v(dtoken), \quad (4.9)$$

where the matrix (i.e., W_v), vectors (i.e., w_{c^*} , w_v , and w_e), and scalars (i.e., b_v and b_c) are trainable model parameters. Incorporating the copy mechanism in our deep learning-based Encoder-Decoder architecture, we get the final loss function for the entire input sequence:

$$L = \frac{1}{T} \sum_{t=0}^T \log p(dtoken_t), \quad (4.10)$$

that is used for our approach to update the parameters θ via backpropagation.

Once our model is trained, we adopt the beam search method to infer the predicted tokens in titles. To be specific, the encoded vectors of a body sequence are sent to the decoder to generate output tokens, and only a certain number of these probable tokens with high scores are retained as a candidate list of the first generated token. Through continuous iteration, the previously retained tokens are re-input into the decoder for the next inferring step. Until the terminator “*EOS*” is encountered or the maximum length of the predicted title is reached, the generated sequence with the highest score is the final output.

4.2 Experimental Setup

4.2.1 Data Preparation

Data for pre-training. We adopted RoBERTa as our encoder, which is pre-trained with five English language corpora [95] of uncompressed text over 160GB, i.e., BOOKCORPUS [209] and English WIKIPEDIA, CC-NEWS [210], OPENWEBTEXT [211], and STORIES [212].

Data for training. We used 333,563 samples collected by Chen et al. [6] as our dataset, which has been split into the training set, validation set, and test set with the ratio of 8:1:1. Each sample represents a bug report, including a pair of “<body,

title>”. The collected dataset has been preprocessed from the raw data containing 922,730 samples by Chen et al. through applying defined rules to filter unnecessary bug reports, which makes the remained data with regular expressions more suitable for studies. In order to avoid the context-specific problem that makes it difficult for our approach to learn the semantic information [213], we replace the human-named tokens (i.e., version number) with special tokens “ $id_1, id_2, \dots, id_n$ ” to preserve the original structure of the input. In addition, we find that the NLTK toolkit [214] cannot separate some tokens, which may cause out-of-vocabulary (unknown tokens) and large-size vocabulary problems. Therefore, we design a tokenizing algorithm based on RoBERTaTokenizer [95], which uses bytes to effectively split special tokens into byte-based subword units [215]. By this means, a smaller byte-level vocabulary is obtained without introducing any unknown tokens when a text sequence is input. For example, the maximum body length in the training set is converted from 300 to 1106 after tokenization, the average and median lengths of bodies and titles are 120.0 and 104.0, 8.2 and 8.0, respectively.

4.2.2 Implementation

In this work, we keep the initial parameter settings of the RoBERTa encoder with a vocabulary of 50,000 tokens. The hidden size is 768, and the Transformer encoder layer is 12. We perform optimization on RoBERTa by using Adam [216] algorithm with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and initial $lr = 5 \times 10^{-5}$ with linear warm-up strategy. We build an 8-layer Transformer decoder with initialized parameters. For training our model, the batch size and training epoch are fixed at 8 and 5, separately. During decoding, we set the beam size to 10. We employ the 5-fold cross-validation on our dataset, and all hyperparameters are adjusted on our validation set, and the average results on our test set of 5-fold cross-validation are reported.

4.2.3 Baselines

Since Seq2Seq approaches have gradually become mainstream, and generally perform better than models with other architectures on text generation tasks [7, 213], we compare our proposed AttSum with two state-of-the-art Seq2seq approaches as the abstractive methods to demonstrate the effectiveness of our approach. In addition, we also design a term-frequency-based approach as the extractive method to show the effectiveness of its performance. In order to ensure a fair comparison, we carefully

tune the parameters involved and set the hyperparameters strictly according to their descriptions to adapt their model to our task.

(1) **iTAPE** Chen et al. [6], for the first time, proposed to automatically generate bug report titles based on the report body and contributed the experimental dataset. Hence, it is regarded as our main baseline. They used an Encoder-Decoder architecture (i.e., one-layer bidirectional LSTM for the encoder and unidirectional LSTM for the decoder) combined with the copy mechanism to conduct title generation, which was implemented by the OpenNMT-py model¹.

(2) **NATS** is an open-source library for abstractive text summarization, which was developed by Shi et al.[7]. This toolkit integrates typical advanced Seq2Seq models with different features, which can be effectively applied to multiple datasets. Therefore, we use the Recurrent Neural Network (RNN)-based Seq2Seq model denoted as ID C10101 in their study that has almost the best performance on the CNN/Daily Mail Dataset [208] as our second baseline. The ID refers to the combination of the attention mechanism, the copy mechanism, the intra-decoder attention, and the coverage mechanism.

(3) **ExtSen** The core of extractive text summarization is to pick out the sentences that best match the summarization [207]. In this study, we take this idea and select only one sentence from the bug report body that has the most overlapping tokens with the target title. Considering the objective of title generation approaches is to maximize the scores of ROUGE and BLEU, we count the frequency of overlapping words and calculate the similarity between each sentence in the bug report body and the target title. Therefore, we name the mentioned simple method ExtSen, which can effectively extract the best result from a model.

4.2.4 Evaluation Metrics

(1) **Automatic Evaluation.** We adopt Recall-Oriented Understudy for Gisting Evaluation (ROUGE) [32] and BBiLingual Evaluation Understudy (BLEU) [33] to evaluate the performance of AttSum and baselines, which are widely used in text summarization and machine translation tasks.

ROUGE [32] is a recall-oriented evaluation metric to evaluate the quality of generated titles. It measures how many the terms in the golden-standard references (original human-written titles) appear in the candidates (model-generated titles). In our study, we employ three ROUGE-family metrics where ROUGE-N (i.e., N=1 and

¹<https://opennmt.net/>

2) scores are calculated with N-gram co-occurrence and the ROUGE-L considers the longest common subsequence.

BLEU [33] is a precision-oriented measure to evaluate the performance of models. Thus, we also use this evaluation metric to automatically measure the generated titles by counting the units in the generated title that occurred in the golden references. In our experiments, we utilize the average BLEU score of N-gram where $N=1,2,3,4$. The usage was also widely adopted, such as in code summarization tasks [217, 218, 207]. For convenience, the composite BLEU is called BLEU in the article.

(2) Human Evaluation.

On the refined dataset. To evaluate the bug report title generation methods more comprehensively, we aim to analyze the text comprehensibility from the human perspective. The human evaluation also acts as a supplementary material to narrow the gap between the existing automatic evaluation metrics and the functionality of the result in reality. In this experiment, we randomly select 200 samples from the test set to make the comparison of approaches fair. Meanwhile, we conduct the statistical analysis to make sure the selected samples are representative, and the results show that the distribution of these samples is consistent with that of the entire dataset (i.e., the mean and median lengths of bodies and titles are 117.1 and 95.5, 6.7 and 6.0, respectively). We invite five volunteers who are not co-authors in this study to inspect the bug report body, then compare the automatically generated titles where the title resources (i.e., which title is generated by which approach) are hidden in advance. Four evaluators are Ph.D. students in computer science, and one evaluator is a master graduate who is engaged in computer science-related work. All of them have over six years of experience in at least two programming languages, and they also have at least three years of studying/working experience in English-speaking countries or regions. Referring to Gao et al.’s recent work [213], we consider the three modalities (accuracy, comprehensiveness, and lucidity) in our case study. Hence, these evaluators are asked to manually score the generated titles, ranging from 1 (weak) to 5 (excellent) across the three modalities with the following grading standards:

- *Accuracy*: It assesses the title’s relevance to the main text and its ability to reflect the critical idea without useless information (e.g., repetition and meaningless words).
- *Comprehensiveness*: The model is supposed to extract input content effectively. Thus, it evaluates whether the title is diverse to contain all necessary information within the limited length.

Chapter 4. AttSum: A Deep Attention-Based Summarization Model for Bug Report Title Generation

- *Lucidity*: It tests the fluency and correctness of the titles. It is expected to be readable by using understandable vocabulary and grammar rules.

On the low-quality dataset. Besides evaluating our approach on the test set, we also investigate how our approach performs when a low-quality bug report is given. It is noteworthy that the dataset described in Section 4.2.1 has been refined and selected from the raw data with approximately 920,000 samples following the instructions of Chen et al.’s [6], so the test set can be used to compare the title generated by our approach with the original reference based on automatic evaluation metrics. For those samples with low-quality titles shown in Table 4.11, it is unlikely to measure the results through automatic evaluation. Thereby, we randomly extract 50 samples with low-quality titles from the raw data, excluding our experimental dataset. The distribution of these selected low-quality pairs (i.e., the mean and median lengths of bodies and titles are 134.3 and 90.0, 7.0 and 6.0, separately) is different from that of the refined dataset but is more consistent with the distribution of the whole low-quality dataset (i.e., the mean and median lengths of bodies and titles are 177.9 and 105.0, 7.0 and 6.0, respectively). This is because in our refined dataset, some bug reports whose body or title is too long or too short are also one of the types of samples that are considered low quality, and therefore are removed from our refined dataset. Then, we ask three of the raters mentioned above to manually decide and annotate the two titles (one is generated by our approach and the other is the original one). Their evaluation consists of two parts, i.e., quantitative scoring and the reasonable explanation in terms of three metrics: fitness, clearness, and willingness, which is consistent with Gao et al.’s work [213].

- *Fitness*: It measures whether the title is reasonable and relevant to summarize the core of the whole bug report.
- *Clearness*: It measures whether the title is complete in grammar and easy to understand with clear expression.
- *Willingness*: It measures whether the title is readable and likely attracts people to handle the specific bug report, which may be effective for us to conduct further interactions.

4.2.5 Statistical Analysis

We employ the t -test [219] to check the significance of the differences between the compared title generation methods [220]. If the obtained p -value is less than 0.05,

4.3 Evaluation and Analysis

Table 4.1 Automatic evaluation of our approach and baselines. F1, P, and R denote the evaluation metrics for F1-score, Precision, and Recall, respectively.

Approaches	ROUGE-1			ROUGE-2			ROUGE-L			BLEU
	F1	P	R	F1	P	R	F1	P	R	
iTAPE[6]	31.03	32.53	31.13	12.69	13.38	12.83	28.62	29.97	28.72	19.56
NATS[7]	24.99	28.35	24.16	9.94	11.46	9.63	24.19	28.52	22.50	14.80
ExtSen	32.32	28.55	46.72	12.07	10.75	18.32	28.23	25.20	40.48	14.96
AttSum	33.41	38.05	31.15	15.78	18.04	14.71	31.69	36.17	29.40	21.06

Table 4.2 Automatic evaluation of our approach and baselines on the long bug report dataset.

Approaches	ROUGE-1			ROUGE-2			ROUGE-L			BLEU
	F1	P	R	F1	P	R	F1	P	R	
iTAPE[6]	27.98	29.71	27.69	11.45	12.40	11.31	26.41	28.26	25.98	16.68
NATS[7]	24.22	27.46	23.54	9.70	11.17	9.42	23.67	28.00	22.18	14.51
ExtSen	31.53	29.44	43.83	11.51	10.78	16.75	27.48	26.00	37.83	14.49
AttSum	30.42	34.61	28.38	14.06	16.05	13.15	28.99	33.12	26.86	19.18

there is a significant difference between the two methods. In addition, the Effect Size (ES) is used to reveal the magnitude of the differences. Considering the ease of calculation and practicality, Cohen’s d (a standardized mean difference) as the most broadly and generally used ES calculation [221] is adopted in our study. For two independent groups, ES (d) can be measured by the standardized difference between two means. The magnitude of the difference is considered negligible ($d < 0.2$), small ($0.2 \leq d < 0.4$), moderate ($0.4 \leq d < 0.8$), and large ($0.8 \leq d$).

4.3 Evaluation and Analysis

We organize the experiment through four Research Questions (RQs), i.e., the titles of the following subsections. The bold values in Table 4.1, 4.2, 4.5, and 4.6 indicate the best performance under automatic or human evaluation; and the bold pair values shown in Table 4.3 and 4.7 represent that AttSum significantly better than the other compared approach in automatic evaluation metrics.

4.3.1 RQ1. How does our approach perform under automatic evaluation?

Methods: To intuitively illustrate the superiority of our approach, Tables 4.1 and 4.2 present the results of automatic evaluation of our approach and aforementioned

Table 4.3 Statistical analysis on the test set and the long bug report test set.

Cohen's <i>d</i> (<i>p</i> -value)		Test set			Long BR test set		
		vs. iTAPE	vs. NATS	vs. ExtSen	vs. iTAPE	vs. NATS	vs. ExtSen
ROUGE-1	F1	4.00E-05 (+Large)	1.21E-03 (+Large)	4.19E-03 (+Large)	5.00E-06 (+Large)	1.43E-04 (+Large)	1.71E-04 (-Large)
	P	2.00E-06 (+Large)	2.66E-04 (+Large)	5.37E-10 (+Large)	2.00E-09 (+Large)	9.00E-06 (+Large)	1.64E-10 (+Large)
	R	9.51E-01 (+Negligible)	5.91E-03 (+Large)	1.13E-04 (-Large)	5.42E-02 (+Large)	3.96E-03 (+Large)	1.57E-11 (-Large)
ROUGE-2	F1	2.67E-08 (+Large)	6.22E-04 (+Large)	5.58E-10 (+Large)	2.31E-07 (+Large)	5.59E-04 (+Large)	3.89E-08 (+Large)
	P	1.77E-08 (+Large)	2.11E-04 (+Large)	1.52E-11 (+Large)	1.30E-09 (+Large)	1.45E-04 (+Large)	7.61E-13 (+Large)
	R	4.00E-06 (+Large)	2.22E-03 (+Large)	3.27E-09 (-Large)	6.00E-05 (+Large)	3.35E-03 (+Large)	4.68E-08 (-Large)
ROUGE-L	F1	4.39E-08 (+Large)	5.21E-04 (+Large)	1.99E-09 (+Large)	5.82E-07 (+Large)	1.86E-04 (+Large)	4.00E-06 (+Large)
	P	6.77E-10 (+Large)	3.10E-05 (+Large)	1.27E-12 (+Large)	1.15E-08 (+Large)	5.00E-06 (+Large)	4.97E-11 (+Large)
	R	1.64E-02 (+Large)	3.60E-03 (+Large)	7.63E-12 (-Large)	1.39E-02 (+Large)	3.21E-03 (+Large)	5.67E-11 (-Large)
BLEU		3.92E-03 (+Large)	1.44E-04 (+Large)	8.96E-08 (+Large)	1.00E-06 (+Large)	3.40E-05 (+Large)	5.03E-09 (+Large)

baselines (i.e., iTAPE, NATS, and ExtSen) on both the entire dataset and the dataset that contains the top 25% long bug reports in the test set for each fold. Each column lists the corresponding results of four models on this evaluation metric, and the best performance is highlighted in boldface. To illustrate the significance of the difference between AttSum and baselines, we calculate the p -value and the effect size of Cohen’s d in Table 4.3. Each cell contains two values, the former one and the latter one within the brackets indicate the p -value and the degree of Cohen’s- d effect size, respectively. Results are shown in bold if they have a large degree of Cohen’s d value and a p -value of 0.05 or less. The positive/negative signs indicate that AttSum has a better/worse performance. In addition, we extract several examples shown in Table 4.4 to perform further analysis.

Results: From Table 4.1, we compare the automatic evaluation results and sum up the following points:

- a) Regarding the ROUGE score, it is clear that our approach delivers outstanding performance compared with the other two abstractive methods on the two sets. On the test set, AttSum outperforms iTAPE, NATS, and ExtSen by **3.4%-34.7%** (ROUGE-1), **24.4%-58.8%** (ROUGE-2), and **10.7%-31.0%** (ROUGE-L) in *F1*; by 17.0%-34.2% (ROUGE-1), 34.9%-67.8% (ROUGE-2), and 20.7%-43.5% (ROUGE-L) in *precision*, respectively. In *recall*, ExtSen performs the best and is superior to AttSum by 19.7%-33.3%. AttSum performs better than iTAPE and NATS by 0.1%-29.0% (ROUGE-1), 14.6%-52.8% (ROUGE-2), and 2.4%-30.7% (ROUGE-L). *Recall* represents the percentage of actual tokens that are correctly identified, and these metrics compare the generated titles to reference titles by measuring the overlapping tokens [222]. Since ExtSen directly extracts the original sentence from the bug report body based on token frequency to best match the title, it is much easier than the abstractive methods (iTAPE, NATS, and AttSum) to contain more overlapping tokens. Regarding the BLEU score, our approach is the most competitive among the four methods. It improves over three baselines on the BLEU by **7.7%-42.3%**. The experimental results on the long BR test set show a similar trend. Compared with iTAPE and NATS, AttSum improves ROUGE in *F1* by 9.1%-22.9% and 22.5%-45.0%, in *precision* by 16.5%-29.4% and 18.3%-43.7%, in *recall* by 2.5%-16.3% and 20.5%-39.5%, respectively. AttSum surpasses ExtSen by 5.5%-22.2% in terms of ROUGE-1 and ROUGE-2 in *F1* but 3.5% below ExtSen (ROUGE-1) in *F1*. For *precision*, AttSum is 17.6%-49.0% better

Chapter 4. AttSum: A Deep Attention-Based Summarization Model for Bug Report Title Generation

than ExtSen, but for *recall*, it is 21.5%-35.3% worse than ExtSen. Considering the BLEU score, AttSum outperforms the three baselines by 15.0%-32.4%.

- b) To illustrate the statistical difference between our approach and other baselines, we calculate the p -value and the effect size of Cohen’s d (Table 4.3). Compared with iTAPE and NATS, AttSum is **significantly better** overall in all evaluation metrics on two sets, except in two situations (i.e., Cohen’s d is at a negligible level, or the p -value is larger than 0.05). ExtSen outperforms AttSum by a large margin in terms of *recall*. On the long BR test set, a high *recall* also contributes to the best $F1$ in ROUGE-1.

Examples: Table 4.4 displays five representative examples that are opted to analyze the performance of the models (i.e., iTAPE, NATS, ExtSen, and AttSum) manually. The detailed observations are listed and explained below.

- a) By order of illustration, iTAPE in the *first* sample generates a title without the command-line program name “youtube-dl” and NATS outputs even an opposite title “downloading all videos from isvaffel2010: s channel” to the original one, which fails to learn the semantic information of the context correctly. Meanwhile, our approach generates a more accurate title “youtube - dl doesn ’ t download videos from isvaffel”, which is basically consistent with the underlined sentence in the report body indicating the primary meaning. In comparison, ExtSen extracts the sentence that can completely describe the main idea of this bug report. However, the length of the extracted title cannot satisfy the rules, it cannot be regarded as a reasonable title.
- b) Another example is the *second* example. Although the condition “Validate=True” is incompletely captured by all models, the title generated by our approach expresses better with fluency to give the primary focus by offering the output owner “functiontransformer” and the description “validate”. The remarkable performance does credit to the capability of our approach for capturing the long-range dependencies.
- c) In addition, the *third* example intuitively demonstrates that our approach has the advantage of understanding the intention of the reporters. Only the title generated by AttSum matches the original request: allow disabling the bootstrap checks. In contrast, the other two abstractive methods select the one of actions conducted by the reporter as the title, respectively. The extractive method only

4.3 Evaluation and Analysis

Table 4.4 Examples of automatically generated titles (four approaches) and the original testing titles.

Body	Titles
<p>(1) When I'm trying to download all the videos from the Youtube user account Isvaffel2010, Youtube-dl just downloads video list pages but doesn't get to downloading the actual videos at all. Other channels I have tried have worked fine, somehow the problem seems to be specific to Isvaffel2010:s channel (although the bug can of course affect some other channels as well). The command I run: <code>youtube-dl-o "/isvaffel2010/(upload_date)s_(title)s_(id)s_(ext)s" --write-description --username xxxxxx --password yyyyyy -f best/18 -v --no-overwrites --restrict-filenames --playlist-end 800 https://www.youtube.com/playlist?list=UUhSBvT-ferzzcAIFFGvMarA. Youtube-dl version: 2014.09.29.2 Python version: Python 2.7.3 My operating system is Xbian release candidate 3 on Raspberry Pi, based on Raspbian. When I run youtube-dl this is the (clipped) output I get http://pastebin.com/cr8k5e2m.</code></p>	<p>Golden: youtube - dl just downloads ridiculously many playlists pages AttSum: youtube - dl doesn't download videos from isvaffel iTAPE: downloading videos from youtube user account does n't work NATS: downloading all videos from isvaffel2010 : s channel ExtSen: When I'm trying to download all the videos from the Youtube user account Isvaffel2010, youtube - dl just downloads video list pages but doesn't get to downloading the actual videos at all</p>
<p>(2) Description Validate ensure that "X" is 2D as it is imposed in any other transformer. However, the output of the "FunctionTransformer" can be 1D if the provided function does not ensure it. I think that we should ensure that the output is 2D since it could be useful when chaining transformer and estimator. "validate=False" will still allow for 1D in this case.</p>	<p>Golden: ensure that the output of functiontransformer is 2d when validate=true AttSum: validate 2d output of functiontransformer iTAPE: validate ensure that x is 2d instead of 2d NATS: functiontransformer should ensure that output is 2d ExtSen: I think that we should ensure that the output is 2D since it could be useful when chaining transformer and estimator</p>
<p>(3) When running elasticsearch in docker, in a development environment or when running builds, I need to have elasticsearch bind to 0.0.0.0 and/or publish a container hostname. This has the undesirable side-effect of enforcing bootstrap checks.I understand the benefit of these bootstrap checks in production, but I question the assumption that you must be in production if you're binding to a non-local address, especially given the prevalence of docker. Having to modify the host to set <code>vm.max_map_count</code>, for example, severely limits the portability of my test environment for no real reason. It would be useful to be able to set a configuration setting explicitly indicating I am not in production and that the bootstrap checks are therefore unnecessary, rather than binding the assumption of development vs production to the hostname to which we choose to bind.</p>	<p>Golden: allow disabling bootstrap checks explicitly AttSum: make it possible to disable bootstrap checks iTAPE: publish bootstrap checks to set vm . max_map_count NATS: elasticsearch bind to 0.0.0.0 and / or publish a container hostname ExtSen: this has the undesirable side-effect of enforcing bootstrap checks</p>
<p>(4) 5.0.0-beta Bulk index request with 1 concurrency error returns no version info (for the one item) as expected. POST http://127.0.0.1:9200/_bulk HTTP/1.1 Accept: application/json Content-Type: application/json Host: 127.0.0.1:9200 Content-Length: 647 ... version conflict, current version [2] is different than the one provided [1] ... bulk delete request for items that don't exist return version "1". Shouldn't these be consistent and not return a version as these documents don't exist?</p>	<p>Golden: inconsistency with returned version numbers on requests with errors AttSum: bulk delete request with 1 concurrency error returns no version iTAPE: bulk index request does n't return version info NATS: bulk index request returns no version info ExtSen: 5.0.0-beta Bulk index request with 1 concurrency error returns no version info (for the one item) as expected</p>
<p>(5) The introduction of Web Flux support in 2.0 will mean that we have a second kind of web application. As described in 8017, this makes "setWebEnvironment(boolean)" on "SpringApplication" insufficient. I think we need an enum. In 1.5, it'd only have two values: NONE and SERVLET. In 2.0 we can add REACTIVE or WEB_FLUX or whatever we want to call it. We'd then deprecate "setWebEnvironment(boolean)" in favour of a new method that takes the new enum. We already have a "WebEnvironment" enum in the testing support with a different meaning so I think we may need another name. I wonder if it could be "WebApplication" with a corresponding "XsetWebApplication(WebApplication)" method on "SpringApplication". "WebApplication" ties in quite nicely with "@ConditionalOnWebApplication" and could also complement more specific variants of that condition in 2.0 such as "@ConditionalOnServletWebApplication" and "@ConditionalOnReactiveWebApplication".</p>	<p>Golden: introduce an enum for web environment on springapplication AttSum: rename webapplication . setwebenvironment (boolean) to iTAPE: deprecate set webenvironment (boolean) in springapplication NATS: web flux support for web flux support ExtSen: As described in 8017, this makes setWebEnvironment(boolean) on SpringApplication insufficient.</p>

Chapter 4. AttSum: A Deep Attention-Based Summarization Model for Bug Report Title Generation

lists the consequence. The information focusing on one point is too specific to help developers identify the root of the problem.

- d) Sometimes, there is an unavoidable gap in the ability to conclude statements between humans and machines, even for the extractive approach. In the *fourth* example, all models only show one error case “no returned version”, and miss the case of “version conflict”. We will handle such problems in our further study, e.g., investigate how to interpret each key point and draw a conclusion.
- e) The proposed model might be disrupted by a large amount of the rare terms in the original text body. In the fifth example, none of the abstractive machines manages to capture the problem the reporters try to solve. There is even repetition in the titles from the NATS model. Apart from that, the application environment “SpringApplication” is missing in all generated titles. On the contrary, they focus on the solutions like “deprecate setWebEnvironment(boolean)” (iTape) and provide a new name “WebApplication=” (AttSum). Thus, it might be hard for the developers to infer where the bug is. In this case, the extracted sentence of ExtSen does not fully describe the problem and has an unclear referent.

Answer to RQ1: AttSum is superior to iTape and NATS on both ROUGE and BLEU evaluation metrics; Even ExtSen performs well in terms of *recall*, the extracted titles are usually still unsatisfying.

4.3.2 RQ2. How does our approach perform under human evaluation?

Methods: (1) We summarize 200 samples with five titles (i.e., four generated and one original) and 50 low-quality samples with two titles (i.e., the generated title by AttSum and the original one), then distribute them to evaluators. As we mentioned in Section 4.2.4, for the former evaluation, evaluators need to score the four generated titles across *Accuracy*, *Comprehensiveness*, and *Lucidity* modalities. We inform evaluators in advance that the higher the score, the better the result. In each modality, a score lower/higher than 3 is considered poor/desired performance, and a score equal to 3 is medium performance. In addition to this, we perform a statistical analysis of these evaluation results on the 200 samples, including effect size and *p*-value.

(2) For each pairwise comparison of low-quality samples, we collect the selections from three raters and calculate which title was better on each assessment dimension (i.e., *Fitness*, *Clearness*, and *Willingness*) regarding the percentage of choices. For example, if two raters vote Candidate 1 and another one votes Candidate 2, the final result should be that Candidate 1 is the winner. If all three raters make different choices, the conclusion should be indistinguishable.

(3) To explore the degree of agreement of evaluators, we follow Hu et al. [222] and compute the Kendall Correlation τ and Pearson correlation r to evaluate the consistency among them. In Tables 4.9 and 4.10, we calculate the correlation values for the two human assessments separately. Based on Hinkle et al.’s [223] scheme, the correlation is considered negligible ($\tau/r < 0.3$), low ($0.3 \leq \tau/r < 0.5$), moderate ($0.5 \leq \tau/r < 0.7$), high ($0.7 \leq \tau/r < 0.9$), and very high ($0.9 \leq \tau/r \leq 1$).

Results: (1) The collected 200 feedbacks and the top 25% long bug reports selected from them (i.e., 50 feedback) are considered as two evaluation sets, and we then calculate the average score and quality distribution for each modality among the four approaches. The statistical results are illustrated in Tables 4.5 and 4.6.

- On the 200-feedback set, AttSum is much better than the three baselines, outperforming them in *Accuracy*, *Comprehensiveness*, and *Lucidity* on average by 3.5%-57.5%, 1.9%-52.6%, and 1.9%-5.9%, separately. Similarly, our approach achieves 5.6%-55.8%, 0.5%-49.8%, and 0.5%-4.5% higher average scores than the three baselines on the three modalities on the 50-feedback set. In most cases, AttSum has the smallest percentage of poor performance and the largest percentage of good performance. The superior performance derives from the caption of core information and complex semantics.
- Table 4.7 lists both effect size and p -value between AttSum and the baselines. In the three modalities, our approach is slightly better than iTAPE and NATS. While ExtSen performs well on automatic evaluation, it performs poorly on human evaluation, especially in terms of *Accuracy* and *Comprehensiveness*. Compared with AttSum, the calculated p -values are much smaller than 0.05, and the Cohen’s d values are at a large degree.

(2): Table 4.8 displays the comprehensive results of the manual evaluation on the 50-long bug report set. From this table, we can see that:

- In terms of all assessment metrics, the titles generated by our approach beat the human-written titles with poor quality. It shows the capability of our approach to generating clearer and more reasonable titles.

Table 4.5 Human evaluation of our approach and baselines.

Approaches	Accuracy			Comprehensiveness			Lucidity		
	Average	Poor	Medium	Good	Average	Poor	Medium	Good	Average
iTAPE	3.69	15.0%	28.5%	56.5%	3.76	11.5%	26.5%	62.0%	4.25
NATS	3.54	20.0%	26.5%	53.5%	3.55	16.5%	31.0%	52.5%	4.09
ExtSen	2.43	53.5%	25.0%	21.5%	2.51	50.5%	33.5%	16.0%	4.19
AttSum	3.82	13.5%	25.0%	61.5%	3.83	11.5%	26.5%	62.0%	4.33
						5.0%	11.5%	83.5%	

Table 4.6 Human evaluation of our approach and baselines on the long bug report dataset.

Approaches	Accuracy			Comprehensiveness			Lucidity		
	Average	Poor	Medium	Good	Average	Poor	Medium	Good	Average
iTAPE	3.58	14.0%	32.0%	54.0%	3.74	12.0%	26.0%	62.0%	4.20
NATS	3.44	22.0%	24.0%	54.0%	3.58	18.0%	28.0%	50.0%	4.00
ExtSen	2.43	54.0%	28.0%	18.0%	2.51	42.0%	46.0%	12.0%	4.04
AttSum	3.78	10.0%	28.0%	62.0%	3.76	12.0%	24.0%	64.0%	4.22
						4.0%	18.0%	78.0%	

Chapter 4. AttSum: A Deep Attention-Based Summarization Model for Bug Report Title Generation

Table 4.7 Statistical analysis on the 200 samples.

Effect size Cohen’s d (p -value)			
Modality	vs. iTAPE	vs. NATS	vs. ExtSen
Accuracy	2.58E-01 (+Negligible)	1.85E-02 (+Small)	8.87E-26 (+Large)
Comprehensiveness	6.94E-01 (+Negligible)	1.82E-02 (+Small)	5.77E-31 (+Large)
Lucidity	3.83E-01 (+Negligible)	1.22E-02 (+Small)	3.12E-01 (+Negligible)

Table 4.8 Manual evaluation on bug reports with low-quality titles.

Generated vs Original			
Modality	Win(%)	Lose(%)	Indistinguishable(%)
Fitness	66	22	12
Clearness	62	20	18
Willingness	58	22	20

- Our approach performs exceptionally well on the *Fitness* metric, generating even 3 times as many suitable titles as human-written ones. It indicates that the titles generated by our approach are more likely to be relevant and grasp the main point of the bug reports.

Table 4.9 The average Kendall Correlation (τ) and Pearson Correlation (r) among evaluators on the test set.

Modality	Accuracy		Comprehensiveness		Lucidity	
	τ	r	τ	r	τ	r
iTAPE	0.80	0.88	0.73	0.77	0.66	0.72
NATS	0.80	0.84	0.78	0.82	0.69	0.71
ExtSen	0.73	0.77	0.66	0.72	0.80	0.84
AttSum	0.75	0.76	0.72	0.75	0.68	0.77

Table 4.10 The average Kendall Correlation (τ) and Pearson Correlation (r) among evaluators on the low-quality test set.

Modality	Fitness		Clearness		Willingness	
	τ	r	τ	r	τ	r
Title Generation	0.71	0.76	0.53	0.59	0.52	0.59

(3) When evaluating the performances of four approaches on the test set (Table 4.9), raters have high Pearson correlations ($0.7 \leq r < 0.9$) in evaluating three modalities for iTAPE, NATS, and AttSum. In addition to their moderate correlations ($0.5 \leq \tau < 0.7$) on *Lucidity*, they still have high Kendall correlations on both *Accuracy* and *Comprehensiveness*. When measuring ExtSen’s performance, evaluators have high two kinds of correlations on all modalities except the moderate Kendall correlations on

Comprehensiveness. In summary, each evaluator rates all approaches very closely on three modalities. Concerning the assessment of the low-quality test set (Table 4.10), evaluators have both high Kendall and Pearson correlations ($0.7 \leq \tau/r < 0.9$) in assessing *Fitness*, and moderate correlations ($0.5 \leq \tau/r < 0.7$) in assessing *Clearness* and *Willingness*. The results denote those evaluators have highly consistent evaluations, and the titles generated by AttSum have a remarkable advantage in *Fitness* compared with the original titles. All in all, for the two types of assessments, the raters are mostly in high agreement or at least moderate agreement (according to Kendall and Pearson correlations) on the different modalities. Consequently, in the absence of any apparent conflict, the inter-rater reliability is acceptable.

Table 4.11 Examples of bug reports with low-quality titles.

Body	Titles
(1) Hello, I know how to support <code>sass</code> files using webpack, but I am little confused if/how that is supported in <code>gatsbyjs</code> as webpack config seems buried in source of gatsby. Thanks in advance for help.	Human: <code>Sass</code> support? AttSum: support for <code>sass</code> files in <code>gatsbyjs</code>
(2) Can I know using MPAndroidChart, it is possible to draw <code>two real time functions on the same line chart</code> . If, yes can you please help me. Thanks.	Human: Not an issue but a question! AttSum: <code>two real time functions on the same line chart</code>

Examples: We give two examples shown in Table 4.11 to demonstrate the performance of our approach in practice visually. One of the reasons for the poor-quality titles is that the content of the report is not accurately and clearly summarized, such as insufficient information (e.g., the *first* example) and too broad description without mentioning the actual problem (e.g., the *second* example) in human-written titles. Instead, in both cases, our approach is able to catch the keywords that even rarely appear (e.g., “sass” and “gatsbyjs” in the first case) and capture the primary intention of the report (e.g., in the second case).

Answer to RQ2: AttSum outperforms two abstractive approaches and overwhelms the extractive approach across all modalities (i.e., *Accuracy*, *Comprehensiveness*, and *Lucidity*); It is able to significantly improve the low-quality titles on *Fitness*, *Clearness*, and *Willingness*.

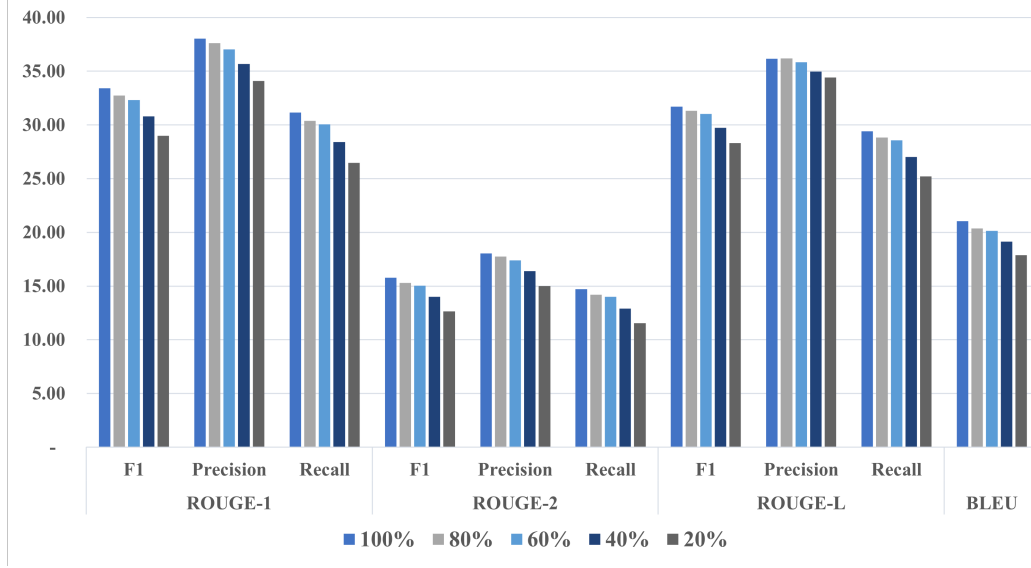


Fig. 4.2 The effect of different ratios of training data on AttSum.

4.3.3 RQ3. How does the size of the training set influence the performance of AttSum?

Methods: The entire dataset used in our study has been split into training/validation sets and test set with a ratio of 9:1. To explore the impact of the size of the training set on AttSum, we perform the automatic evaluation on different proportions of the training set (i.e., 80%, 60%, 40%, and 20%) where the validation set is also cut down by the same ratio. In Figure 4.2, each color represents a kind of training set size, and each bar corresponds to an automatic evaluation value (i.e., ROUGE and BLEU scores).

Results: Overall, the performance on each evaluation metric maintains a steady decline as the training set size decreases. In detail, when taking 80% and 60% of the original training data as the training set size, the performance of AttSum is only reduced by 2.1% and 3.4% on average on all evaluation metrics, respectively, and the Standard Deviations (SD) are 0.01. The performance of the AttSum drops by 9.0% at 40% of the training set, and the SD becomes larger, i.e., 0.03. Compared to the former three training set size, the average performance declines apparently (16.9%) when the scale is 20%, and its SD is even double (0.06) that of the 40% training set. Our results imply that the performance of the AttSum may suffer to a relatively large extent only if the training and validation sets are significantly reduced in size to 20% of their original sizes (i.e., with a ratio of 1.8:1 to the test set). 40% of the original training/validation set size does not seriously impair the performance of the model (i.e., with a ratio of

3.6:1 to the test set). On the basis of the results, we recommend that the data size for training and tuning should not be less than three times the test set (i.e., the ratio of training/validation set and test set should be greater than 3:1). It also indicates that the AttSum is robust enough even though the size of the training and validation sets is very limited.

Answer to RQ3: AttSum has good robustness even when the training/validation set size is reduced to 40% of the original size.

Table 4.12 Ablation studies of AttSum.

Strategies		M_1	M_2	M_3	M_4	M_5
ROUGE-1	F1	32.51	32.38	33.05	32.52	33.41
	P	36.97	36.86	37.43	37.45	38.05
	R	30.38	30.22	30.97	30.08	31.15
ROUGE-2	F1	15.39	15.38	15.47	14.89	15.78
	P	17.60	17.61	17.64	17.25	18.04
	R	14.40	14.37	14.50	13.80	14.71
ROUGE-L	F1	31.42	31.40	31.53	30.89	31.69
	P	36.42	36.55	35.84	35.70	36.17
	R	28.91	28.80	29.34	28.41	29.40
BLEU		20.53	20.45	20.66	20.32	21.06

4.3.4 RQ4. How do the feature parts influence the performance of AttSum?

Methods: (1) We conduct the ablation analysis (Table 4.12) to examine the effectiveness of our encoder, decoder, and the copy mechanism using automatic evaluation, and we compare AttSum (M_5) with four incomplete variants ($M_1 - M_4$). Model M_1 combines a RoBERTa encoder and a one-layer LSTM decoder. Model M_2 combines a RoBERTa encoder and a one-layer GRU decoder. Model M_3 combines a BERT encoder and a stacked Transformer decoder. Model M_4 removes the copy mechanism from our approach.

(2) To evaluate whether incorporating a copy mechanism is able to statistically enhance the performance of our approach, we also calculate the effect size and p -value in Table 4.13. In addition, two examples are enumerated for further explanation.

Results: (1) From the results shown in Table 4.12, we can see that the four model parts adopted in our approach can enhance the performance in terms of evaluation metrics.

Chapter 4. AttSum: A Deep Attention-Based Summarization Model for Bug Report Title Generation

Table 4.13 Statistical analysis of the effectiveness of the copy mechanism.

Cohen's d (p -value)		AttSum without the copy mechanism
ROUGE-1	F1	7.24E-02 (+Large)
	P	1.54E-01 (+Large)
	R	6.32E-02 (+Large)
ROUGE-2	F1	3.96E-03 (+Large)
	P	7.31E-03 (+Large)
	R	6.77E-03 (+Large)
ROUGE-L	F1	3.64E-03 (+Large)
	P	6.15E-02 (+Large)
	R	1.43E-02 (+Large)
BLEU		1.91E-01 (+Large)

As for ROUGE metric, AttSum outperforms four models by **1.1%-3.2%** (ROUGE-1), **2.0%-6.0%** (ROUGE-2), and **0.5%-2.6%** (ROUGE-L) in *F1*; by 0.6%-3.6% (ROUGE-1), 1.4%-6.6% (ROUGE-2), and 0.2%-3.5% (ROUGE-L) in *recall*, respectively. In *precision*, AttSum performs better than M1-M4 by 1.6%-3.2% (ROUGE-1) and 2.3%-4.6% (ROUGE-2) while M1 and M2 is lightly superior to AttSum by 0.7%-1.1% (ROUGE-L). As for the BLEU metric, the score is improved by **1.9%-3.6%** when compared with the other four models.

(2): The copy mechanism can bring substantive significance (large effect size) and statistical significance (p -values less than 0.05) to the performance of our approach on the *F1* metrics. Although statistical significance could not be observed on some evaluation metrics, substantial significance is always present.

Table 4.14 Examples of generated titles by AttSum with or without copy mechanism.

Body	Titles
(1) Hi all ! i can not using "extracttextplugin" for css and sass i set "webpack.config.js" phofcode and " app.component.ts" phofcode and i run it still error uncaught error : expected "styles" to be an array of strings . zone.js:129 help me , thank so much	Golden: can ' t load sass and css With copy: can not use extracttextplugin for css and sass Without copy: can not use "extracttextplugin" for css and s
(2) the compilation of rustdoc crate for target "aarch64-linux-android" failed to compile. it seems the "os" module in crate isn't defined for "target_os=\ android \ ". But as I don't have this platform, I can't check how implement it.	Golden: rustdoc compilation error for aarch64 - linux - android With copy: rustdoc fails to compile for aarch64 - linux - android Without copy: failed to compile rustdoc for android

Examples: Table 4.14 presents some examples to indicate the effectiveness of employing the copy mechanism. All the displayed cases have two emphasizing points highlighted with different colors in the original report body. The text marked with the corresponding color in titles means that this crucial point is captured.

Apparently, the bug report in the *first* example states the problem at the beginning, but the golden reference did not declare it clearly. In comparison, our approach captures the rare terms “extracttextplugin” and “css and sass” simultaneously, while the second phrase cannot be captured completely without a copy mechanism. Another case is the *second* example, and the reporter sought advice on a specific compilation error. The original title detailed the “aarch64-linux-android” module, and our approach with the copy mechanism also extracts the keyword. In comparison with them, our approach without the copy mechanism is unable to capture the key point completely.

Answer to RQ4: The RoBERTa encoder, the stacked Transformer decoder, and the copy mechanism have a positive effect on the performance of our approach based on automatic evaluation; adding the copy mechanism could bring about a significant difference in statistical and human analysis to a certain extent.

4.4 Discussion

4.4.1 Automated Metrics for Evaluating Title Quality

A good-quality bug report title is able to help the software maintainers quickly catch issues and fix them in time. As a mandatory section, the title writing suggestions are included in the guidelines published by the bug trackers for reporters to follow, e.g., “include an informative, scannable title; try and limit the number of words as much as possible”, “a good summary should quickly and uniquely identify a bug report; it should explain the problem, not your suggested solution”, and “your title should serve as a concise summary of what the bug is.” However, even though there are now some specific instructions to help people write reports, it is inevitable that some bug reporters may be unfamiliar with bugs and cannot summarize the problem in an accurate and general manner.

Hence, we recommend AttSum automatically generate a one-sentence summary for reference. In order to follow existing specifications as much as possible, this abstract approach extracts key information by learning semantics from the bug report body, and follows the rules of high-quality titles by limiting the length of the generated titles, duplicating rare terms, etc. In Section 4.3, we combine automatic metrics with human judgment to evaluate the performance of different approaches, and also calculate the agreement of rater correlations (i.e., Kendall and Pearson correlations). Although we tried our best to provide a fair comparison, there is no evidence that these

metrics correlate with human judgment (e.g., ExtSen has acceptable performance on metrics, but worst performance on human judgment). Hu et al. [222] also pointed out that for code documentation generation tasks, these automatic metrics are not sufficient substitutes for human evaluation. In future research, we still need to develop specialized automated evaluation metrics for bug report title generation tasks, which are more closely related to human evaluation metrics.

4.4.2 Is AttSum Good Enough

The *F1* scores for all approaches shown in Table 4.1 may not be as high as we expected, which brings us a question of whether these methods are really helpful in guiding reporters to write titles. To deeply explore the effectiveness of AttSum, we selected some real-world instances for analysis and conducted many human evaluation experiments in Section 4.3.2. In Table 4.4, we discussed the generated titles and found that, although ExtSen also has reliable *F1* scores among the four approaches because of the high degree of overlapping tokens, these titles are often considered unqualified by human evaluation (e.g., too long in length, or failing to capture the main idea of the bug report). Furthermore, even if the tokens generated by abstractive approaches are very similar to the tokens that appeared in the original titles, automatic metrics do not count them. Hence, evaluation metrics are not sufficient for us to judge the quality of the generated titles. We invited several evaluators to manually score the generated titles without knowing which title was generated by which approach. The results displayed in Tables 4.5 and 4.6 indicate that AttSum performs the best in terms of three modalities, and Table 4.8 implies that AttSum effectively generates titles of decent quality, which motivates us to apply this approach in the future to generate drafts that reporters can use as a reference to revise and refine.

4.4.3 Error Analysis and Suggestions

To intuitively compare the performance of different approaches, we provided five examples in Table 4.4 and explained them in detail in Section 4.3.1. The *fourth* example and the *fifth* example indicate that the gap between humans and machines is unavoidable. Due to a large number of rare terms in the text body, it is hard for abstractive approaches to capture the problem raised in the original title. A potential explanation is that AttSum puts more focus on the semantics and long-range dependency capture, and even incorporating with a copy mechanism, it might be difficult to choose the appropriate one [205] when multiple rare terms appear. In

the *fourth* example, all models fail to capture all key points, they miss the error case “version conflict” that is embedded in code snippets and difficult to identify. For the *fifth* example, instead of the specific environment “SpringApplication”, our approach captured another term “WebApplication” to demonstrate a possible solution rather than the problem statement. Nonetheless, AttSum and iTAPE came up with the closest title compared to the other two approaches. Hence, adopting AttSum to generate titles is a good choice to avoid some biases like the given examples. Therefore, we suggest that reporters can consider this approach as an automatic generation tool for reference, and occasionally they may need to manually modify the generated titles. Moreover, we will explore how to accurately capture rare terms in the future.

4.5 Threats to Validity

4.5.1 Internal Validity

Threats to internal validity concern the relationship between our implementation and the corresponding output. To try our best to avoid errors, we have thoroughly checked and tested the code of our approach implementation. To have a fair comparison, we carefully went through the experimental setup of our baselines and tuned the parameters that were then adopted in their best-performing settings. Hence, the results of our automatic evaluation shown in Section 4.3 can be convincing.

4.5.2 External Validity

Threats of this type are related to the generalizability of our dataset. We adopted the collected dataset from Chen et al. [6], which has been preprocessed by several rules, e.g., the non-textual information in bug reports should be replaced with regular expression. Considering the model architecture of the baselines, these non-textual data may increase the difficulty of model learning. Nevertheless, for our approach with a more powerful deep learning capability, this information may be helpful for a deeper investigation of the bug reports [6]. Furthermore, whether AttSum can be applied in a cross-project prediction scenario is also a concern to us. We did not conduct pure cross-project experiments (i.e., the training set and the test set were from completely different repositories), but the data used for training, validation, and testing were randomly selected in our experiment. Consequently, AttSum heavily utilized other repositories as the training and validation data for training and tuning when

Chapter 4. AttSum: A Deep Attention-Based Summarization Model for Bug Report Title Generation

generating titles from bug report bodies from a repository. Additionally, according to our experimental results illustrated in Figure 4.2, despite the ratio of training/validation set and test set dropping nearly threefold (i.e., from 9:1 to 3.6:1), the results only reduced by 9.0%. Since our dataset contains many repositories and AttSum has the ability to learn the different features of them, it is possible to be utilized for other project research even if the dataset size is limited. In short, our approach not only offers cross-project potential but also expresses its robustness.

4.6 Chapter Summary

This chapter has introduced AttSum, a deep attention-based summarization model that combines an Encoder-Decoder model with a copy mechanism, which is proposed to improve the performance of bug report title generation. The model architecture effectively captures the semantic information and long-range dependencies from report bodies regardless of distance among terms. The copy mechanism is incorporated to extract the necessary rare terms into the generated titles selectively. Extensive experiments were conducted with 333,563 pairs of real-world bug reports. The automatic evaluation results showed that AttSum is superior to the state-of-the-art approaches, consistent with the conclusion drawn from human evaluation. Furthermore, we survey the impact of the training dataset size on AttSum and conduct a practical analysis on the low-quality pairs, and the results indicate that AttSum is robust enough and can significantly enhance the quality of original titles.

Chapter 5

AdaLog: A Semi-supervised Approach for Industrial Anomaly Detection via Self-adaptive Clustering

With the rapid development of the Industrial Internet of Things (IIoT), log-based anomaly detection has become vital for smart industrial construction that has prompted many researchers to contribute. To detect anomalies based on log data, semi-supervised approaches stand out from supervised and unsupervised approaches because they only require a portion of labeled data and are relatively stable. However, the state-of-the-art semi-supervised approaches still suffer from two main problems: manual parameter setting and unsatisfactory performance with high false positives. We propose AdaLog, an integrated semi-supervised approach based on self-adaptive clustering, for industrial anomaly detection. In particular, the clustering step performs automatic label probability estimation by distinguishing twelve situations so that the label probability of each unlabeled data can be carefully calculated, leading to high accuracy. In addition, AdaLog employs a pre-trained model to learn contextual information comprehensively and a transformer-based model to detect anomalies efficiently. To alleviate class imbalance, an undersampling method is incorporated. The results on three popular datasets demonstrate that AdaLog significantly outperforms three state-of-the-art semi-supervised approaches by 17.8%–2,489.8% on average in terms of F1-score, and is even superior to two supervised approaches in most cases with average improvements of 10.9%–23.8%. This work was published in IEEE Transactions on Industrial Informatics in 2023 [48].

Chapter 5. AdaLog: A Semi-supervised Approach for Industrial Anomaly Detection via Self-adaptive Clustering

The following sections of this chapter are structured as follows: Section 5.1 introduces the novel approach AdaLog. Section 5.2 outlines the experimental settings, while Section 5.3 presents the results and discussion. Finally, Section 5.4 provides a summary of the chapter’s work.

5.1 Methodology

The workflow of AdaLog is illustrated in Figure 5.1, which contains four modules: data preprocessing, semantic embedding and undersampling, self-adaptive clustering, and classification model building.

AdaLog 1) preprocesses the log messages in a standard format (Section 5.1.1). The processed log data is 2) tokenized and transformed into vector representations by a pre-trained BERT model (Section 5.1.2). After that, the training log messages are grouped into log sequences with/without ground truth labels (i.e., labeled or unlabeled), and then are processed through undersampling. 3) The remaining log sequences are collected together for automatic clustering and label probability estimation (Section 5.1.3). AdaLog 4) builds the Transformer-based model for classification (Section 5.1.4) and trains it using the training data with estimated labels. Once deployed, AdaLog first generates log vectors by steps 1) and 2) given a test log sequence. The vectors are then fed into the trained model, from which the predicted output will be normal or abnormal.

5.1.1 Data Preprocessing

The first step of AdaLog is preprocessing the raw log messages. To avoid the errors caused by log parsing (e.g., Drain, Spell, AEL, and IPLoM) due to the lack of semantic information and semantic misunderstanding [11], we keep the full information of each log message. The raw log messages are tokenized into word-based tokens and then separated by common delimiters (e.g., commas, semicolons, and white spaces). Inspired by NeuralLog [11], AdaLog removes all non-character tokens, such as numbers, operators, and punctuations, which preserves most of the informative content. In addition, AdaLog converts the capital letters to lower letters to make the overall process case-insensitive. An example of log message preprocessing is shown in Table 5.1.

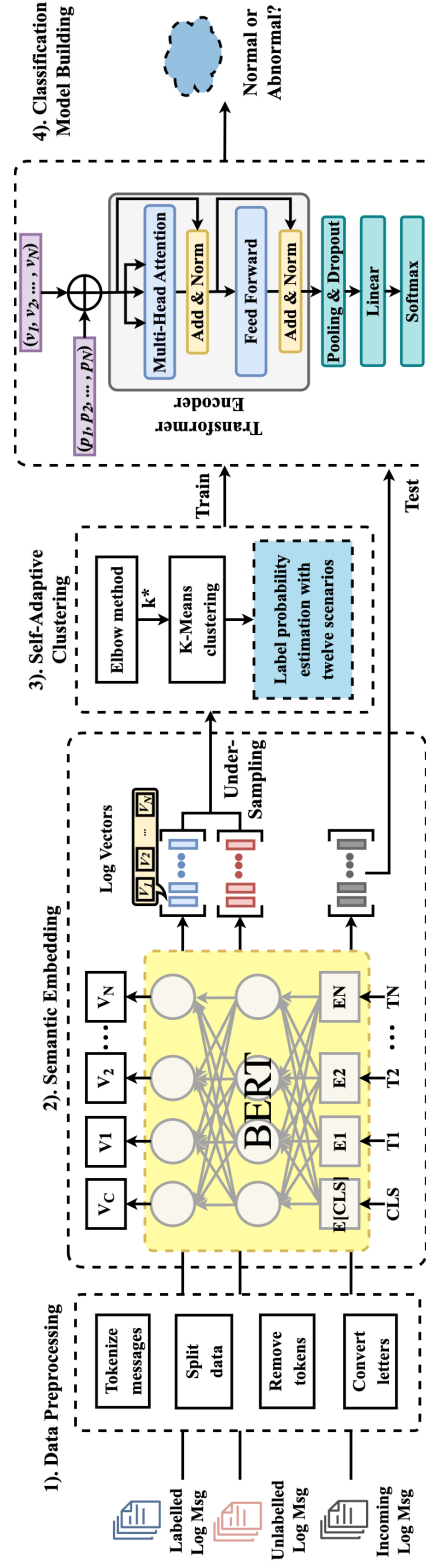


Fig. 5.1 The four-phase framework of AdaLog.

Table 5.1 An example of log message preprocessing on BGL dataset.

Original: - 1117958437 2005.06.05 R37-M0-N0-C:J13-U01 2005-06-05-01.00.37.726577 R37-M0-N0-C:J13-U01 RAS KERNEL INFO 1 ddr errors(s) detected and corrected on rank 0, symbol 12, bit 3
Preprocessed: ras kernel info ddr errors s detected and corrected on rank symbol bit

5.1.2 Semantic Embedding and Undersampling

To better understand the semantic information of log messages, we adopt a deep learning pre-trained model BERT [20] to extract features and represent log messages as embedding vectors. The architecture of the pre-trained model consists of multi-layers of bidirectional Transformer encoder. Every encoder utilizes the self-attention mechanism to determine the terms they should pay more attention on and draw global dependencies between inputs and outputs. Hence, each input log message from the training set is passed through the first-layer encoder to generate embedding vectors, which serve as the input of the next layer of the encoder. The word embeddings generated by the last-layer encoder of BERT are used for the next step. In this way, each log message can be represented as a fix-length vector, i.e., $V = \{v_1, v_2, \dots, v_N\}$, where N is the number of tokens in each log message. Subsequently, the log messages are grouped into log sequences by session or fixed windows.

As we mentioned in Section 1.3, class imbalance exists in the log-based anomaly detection datasets. Therefore, we employ an undersampling method [224] to remove a certain proportion of normal log sequences to alleviate this problem. We did not use oversampling since oversampling may lead to overfitting during the model construction process, especially when abnormal data is very limited. To handle the varying degrees of imbalance in the datasets, we empirically design the following heuristic rule: use an undersampling ratio range and then specify three-quarters of the undersampling range as the specific undersampling ratio to conduct undersampling. Finally, we remove normal log sequences from the training dataset according to the order of the collected data. The remaining normal log sequences and all abnormal log sequences are sent to the clustering stage as training data.

5.1.3 Self-adaptive Clustering

Typically, the training dataset of semi-supervised approaches consists of labeled and unlabeled data. To utilize labeled data, semi-supervised approaches employ clustering

algorithms to form groups of similar samples, based on which we can estimate the label probability of the unlabeled data according to specially-designed different situations.

Existing clustering methods can be classified into four categories. Since the hierarchical-based clustering technique has the drawback of high time complexity, the density-based and grid-based are sensitive to parameter selection, AdaLog uses k -means [174], a widely-used partition-based method to cluster the samples. The principle of this method is to optimize the squared error distortion between the samples and the centroids to minimize the within-cluster variance by adjusting the centroids.

(1) K-Means Clustering with Elbow Method. As explained above, we choose the effective and straightforward method k -means for rough clustering. To ensure the most suitable k (i.e., the number of clusters), we use the Elbow method [172] to compute the recommended k^* . The Elbow method is centered around the computation of the sum of squared errors (SSE). The SSE measures the sum of the squared distances between each data sample and its corresponding cluster centroid, which represents the within-cluster variability. The SSE can be regarded as the clustering error of all samples, and serves as an indicator of the clustering quality. When the number of clusters, denoted by k , is smaller than the true number of clusters, increasing k will lead to a significant decrease in SSE due to the increased degree of aggregation within each cluster. However, as k approaches the true number of clusters, denoted by k^* , the rate of decrease in SSE will begin to plateau, resulting in an elbow-shaped curve. The value of k^* corresponding to the elbow point represents the optimal number of clusters for the given dataset. To compute the SSE for each value of k within a range $K = (0, 50)$, we sum the squared distance as shown in the following equation:

$$SSE_k = \sum_{i=1}^k \sum_{p \in C_i} |p - m_i|^2. \quad (5.1)$$

where C_i represents the i -th cluster, p denotes a sample point belonging to C_i , and m_i indicates the centroid of C_i . Based on our empirical findings, we recommend adopting the first two optimal k^* values for subsequent operations, as they may be very close in certain scenarios. For each chosen k^* , we employ k -means to cluster our training data on the embedding space. k -means iteratively computes centroids by calculating the index of the cluster to which each sample is assigned and the distance between each sample and its nearest cluster centroid. When the calculated index of each sample's cluster is its nearest cluster, the iteration stops, and the centroids will no longer change.

Table 5.2 Twelve scenarios of self-adaptive clustering.

Situation		P_{normal}
$S1$. The cluster only has one known category.	$S1.1$ labeled normal samples d_N	$P = 0.99$
	$S1.2$ labeled abnormal samples d_A	$P = 0.01$
$S2$. The cluster does not have any known category. The cluster only has unlabeled samples.		$P = 0.5$
$S3$. The cluster contains two known categories, and $d_N < d_A$.	$S3.1$ $d \leq d_N < d_A$	$P = 1 - (\frac{d_N - d}{d_A - d} * \frac{d_N}{d_A} * 0.5),$ $P \in (0.5, 1)$
	$S3.2$ $d_N \leq d < d_A$	$P = 0.5 + (1 - \frac{d - d_N}{d_A - d}) * (1 - \frac{d_N}{d_A}) * 0.5,$ $P \in (0.5, 1)$
	$S3.2.1$ $d - d_N \leq d_A - d$	
	$S3.2.2$ $d - d_N > d_A - d$	$P = 0.5 - (1 - \frac{d - d_N}{d_A - d}) * (1 - \frac{d_N}{d_A}) * 0.5,$ $P \in (0, 0.5)$
	$S3.3$ $d_N < d_A \leq d$	$P = 0.5 - (1 - \frac{d - d_A}{d - d_N}) * (1 - \frac{d_N}{d_A}) * 0.5,$ $P \in (0, 0.5)$
$S4$. The cluster contains two known categories, and $d_A < d_N$.	$S4.1$ $d < d_A < d_N$	$P = 0 + (\frac{d_A - d}{d_N - d}) * \frac{d_A}{d_N} * 0.5,$ $P \in (0, 0.5)$
	$S4.2$ $d_A \leq d < d_N$	$P = 0.5 - (1 - \frac{d - d_A}{d_N - d}) * (1 - \frac{d_A}{d_N}) * 0.5,$ $P \in (0, 0.5)$
	$S4.2.1$ $d - d_A \leq d_N - d$	
	$S4.2.2$ $d - d_A > d_N - d$	$P = 0.5 + (1 - \frac{d_N - d}{d - d_A}) * (1 - \frac{d_A}{d_N}) * 0.5,$ $P \in (0.5, 1)$
	$S4.3$ $d_A < d_N \leq d$	$P = 0.5 + (1 - \frac{d - d_N}{d - d_A}) * \frac{d_A}{d_N} * 0.5,$ $P \in (0.5, 1)$
$S5$. The cluster contains two known categories, and $d_N = d_A$.		$P = 0.5$
P represents the normal probability of each sample, which is calculated by comparing the clustering information of each unlabeled data (d) and labeled data (d_A and d_N).		

In this way, the labeled/unlabeled data has its corresponding cluster under each optimal k^* .

(2) Label Probability Estimation. In this step, our aim is to compute the probability P_{normal} for each unlabeled sample being normal under each k^* value, and the final P_{normal} is obtained by taking the average. Compared to other approaches, AdaLog proposes a novel way to leverage the clustering results and existing labeled samples in the clusters by considering twelve specially-design situations. By doing so, existing labeled data can contribute to the label probability estimation of unlabeled data. We discuss twelve scenarios and demonstrate the computation in Table 5.2. In addition, we keep the $P_{normal} = 1$ for labeled normal samples and $P_{normal} = 0$ for labeled anomalous samples.

The twelve scenarios can be classified as five main situations ($\mathcal{S}1$ - $\mathcal{S}5$) based on which cluster an unlabeled sample belongs to. We calculate the label probability P_{normal} for each unlabeled sample by comparing the distances d , d_A , and d_N .

$$\begin{aligned} d_N &= \frac{1}{N_n} \sum_i^{N_n} d_i, \\ d_A &= \frac{1}{N_a} \sum_j^{N_a} d_j, \end{aligned} \tag{5.2}$$

where d represents the distance from the unlabeled sample to its corresponding cluster centroid. d_N and d_A are the average distances between each labeled normal/anomalous sample in this cluster and the cluster centroid. N_n and N_a indicate the number of labeled normal and abnormal samples, respectively.

The first situation $\mathcal{S}1$ is that the corresponding cluster only contains one known category (normal or abnormal). $\mathcal{S}1.1$ indicates the cluster only has labeled normal samples (i.e., so we have d_N). Therefore, this unlabeled sample is very likely to be normal, and a large probability (P_{normal} is almost closer to 1) is given to this sample. In contrast, $\mathcal{S}1.2$ describes the situation in that only labeled abnormal samples exist in the cluster (i.e., so we have d_A). Accordingly, a small probability (P_{normal} is almost closer to 0) is given to this sample. $\mathcal{S}2$ demonstrates another situation the cluster does not contain any labeled samples. Since there is no preference for the possible estimated categories, we assign a medium probability to the unlabeled sample (i.e., $P_{normal} = 0.5$).

For the $\mathcal{S}3$ and $\mathcal{S}4$ (i.e., the cluster contains two known categories), the average distances d_N and d_A should be considered to decompose the situation. For each

Chapter 5. AdaLog: A Semi-supervised Approach for Industrial Anomaly Detection via Self-adaptive Clustering

sub-situation, we introduce two coefficient parameters P_1 and P_2 for calculating P_{normal} as shown in Table 5.2. P_1 represents the confidence degree in which the sample is more inclined to be normal or abnormal, according to the average distances (i.e., d_N and d_A) from the normal and abnormal samples to the cluster centroid. P_2 refers to the confidence degree that the unlabeled sample is more likely to be normal or abnormal, depending on which category the sample is closer to (i.e., $|d - d_N|$ and $|d - d_A|$). In addition, we utilize constants (i.e., 0.5) to ensure that the P_{normal} is located in the reasonable range. If $d_N < d_A$ (i.e., $\mathcal{S}3$), it demonstrates that the cluster may be more towards the normal category. For $\mathcal{S}3.1$, if the situation satisfies $d < d_N$, the unlabeled sample is closer to the cluster centroid, so it is more likely to be normal (i.e., $P_{normal} \in (0.5, 1)$). The coefficient parameters (P_1 and P_2) expressed as $\frac{d_N}{d_A}$ and $\frac{d_N - d}{d_A - d}$ are used to assess the degree to which the sample tends to be normal. Take the scenario as an example, 1) if the d_N approximates to d_A , it becomes challenging to determine to which category the sample is more likely to belong because known normal and abnormal data are very close in distance within the cluster. Therefore, P_1 as a coefficient cannot excessively bias the sample towards the normal category, even if the sample is closer to the normal category. Similarly, Since the distance between this sample and each category is very close, P_2 does not make much effort to push the sample to one of the categories. Consequently, P_1 and P_2 should be very close to 1, resulting in a more neutral label probability for this sample; 2) In contrast, if the d_A is far away from d_N , P_1 and P_2 should be more biased towards 0, which increases the probability of the sample being normal.

For $\mathcal{S}3.2$, when the sample lies between the two categories (i.e., $d_N \leq d < d_A$), two situations may arise. If $(d - d_N) \leq (d_A - d)$ ($\mathcal{S}3.2.1$), this means that although the sample is likely to be in either of the two categories, the sample is closer to the normal category (i.e., $P_{normal} \in (0.5, 1)$). These two coefficient parameters $(1 - \frac{d_N}{d_A})$ and $(1 - \frac{d - d_N}{d_A - d})$ cause the sample to be shifted in the normal direction; else if $(d - d_N) > (d_A - d)$ ($\mathcal{S}3.2.2$), the sample is more likely to be an anomaly (i.e., $P_{normal} \in (0, 0.5)$), and P_1 and P_2 are formulated as $(1 - \frac{d_N}{d_A})$ and $(1 - \frac{d_A - d}{d - d_N})$ to decide the degree to which the sample is biased like an anomaly. $\mathcal{S}3.3$ indicates that the sample is farther from the cluster centroid and even further than the abnormal category (i.e., $d_N < d_A \leq d$), therefore it is more prone to be an anomaly (i.e., $P_{normal} \in (0, 0.5)$). $(1 - \frac{d_N}{d_A})$ and $(1 - \frac{d - d_A}{d - d_N})$ are used to calculate the degree to which the sample is inclined to the abnormal category.

Similarly, the $\mathcal{S}4$ with $d_A < d_N$ is derived from the same calculation idea of the $\mathcal{S}3$. This situation demonstrates that the cluster is more likely to gravitate toward the

abnormal category due to the fact that known anomalous data are closer to the cluster centroid. The last situation $\mathcal{S}5$ is that $d_N = d_A$. Since we cannot judge which category of the unlabeled sample is more likely to belong, we assign the $P = 0.5$ to it under this condition. It can be regarded as a special case of the above cases. Furthermore, we list the range of the calculated P_{normal} for each situation in Table 5.2. It also implies that self-adaptive clustering is reasonable and rigorous.

In this way, unlabeled log sequences can be automatically classified into one of the twelve situations, then their corresponding label probabilities of being normal are calculated. We use $(P_{abnormal} = 1 - P_{normal})$ to represent the probability that a sample is an anomaly. The probability pair $(P_{normal}, P_{abnormal})$ for each log sequence is sent to the next classification model.

5.1.4 Classification Model Building

Based on the training data with ground-truth labels or label probabilities computed by the clustering method, we finally utilize a transformer-based classification model [31] to detect anomalies. Since a log sequence contains many log messages, relative position information should be incorporated. To this end, a sinusoidal encoder [31] used to generate the embedding p_i at position i is added to the corresponding embedding vector v_i , which is fed into the classification model as illustrated in step 4) of Figure 5.1.

The network structure of the transformer encoder solves the limitation of the parallel ability, and better deals with the problem of long dependencies within log sequences. Typically, the transformer encoder stacks several identical layers. Each layer uses multi-head attention and a feed-forward network that contains two fully connected layers, plus layer normalization and residual connection. In this way, the combination of attention scores for each log message is obtained through the fully connected layers [225]. Afterward, the output of this transformer encoder is sequentially connected to a pooling layer, a dropout layer, and a fully connected layer with a softmax function. Since we train with the estimated label probabilities of the log sequences rather than binary labels (i.e., 0 or 1), the transformer-based model will try to optimize the loss between the computed label probabilities and the predicted binary labels. We have such a computation setup for two reasons. The first reason explains why we utilize estimated label probabilities. AdaLog as a semi-supervised method, there is no guarantee that every unlabeled data will be correctly classified by the predictions of the clustering method, and the mislabeled data may bias the training.

Hence, we compute the label probabilities of the data to reduce the noise effect to some extent. The second reason explains why we choose the binary labels predicted by the transformer-based model to compute the loss instead of the probabilistic ones. Owing to the binary classification task, the final prediction result should be normal or abnormal (i.e., 0 or 1). If we use both probabilities to calculate the loss simultaneously, some labels of the data are likely to be neutral, making it difficult to judge the boundary between these two categories.

During the training, the best-trained parameters of this model will be saved for anomaly prediction. If a new log sequence comes, it is preprocessed and embedded before being fed into the trained model. Finally, its predicted label (normal or abnormal) will be generated as output with the above steps.

5.2 Experimental Setup

5.2.1 Datasets

To evaluate AdaLog’s performance in detecting anomalies, we use three widely-used datasets: Hadoop Distributed File System (HDFS) dataset [40], Blue Gene/L supercomputer (BGL) dataset [42]), and Thunderbird dataset [42]. In particular, Thunderbird is an extremely imbalanced dataset, which better reflects anomaly detection in practice. Similar to previous works [11, 13, 12], we group log messages of the HDFS dataset by session windows according to the log’s identifier (i.e., *block_id*). For other datasets (i.e., BGL and Thunderbird), log messages are grouped with fixed *window sizes* (i.e., 20, 100, and 200). In Table 5.3, the number of log messages, the grouping strategies, and the corresponding grouped log sequences with abnormal proportions are presented in order.

5.2.2 Baselines

Recently, log-based anomaly detection as an interesting and novel research problem has attracted many studies proposing models to detect anomalies. Since supervised methods usually have the best performance and our approach is semi-supervised, we choose the state-of-the-art semi-supervised approaches (i.e., DeepLog [8], LogAnomaly [9], and PLELog [13]) and supervised approaches (i.e., LogRobust [10] and NeuralLog [11]) introduced in Section 2.3.1 as baselines.

Table 5.3 Statistics of three datasets.

Datasets	Category	Messages	Grouping	Log Sequences	Training Data (anomaly)	Test Data (anomaly)
HDFS	Distributed system	11,175,629	session	575,061	460,048 (2.9%)	115,013 (2.9%)
BGL	SuperComputer	4,747,963	ws=20	237,397	189,918 (9.1%)	47,479 (6.3%)
			ws=100	47,478	37,983 (10.5%)	9,495 (8.6%)
			ws=200	23,738	18,991 (11.6%)	4,747 (10.1%)
			ws=20	499,998	399,999 (0.4%)	99,999 (0.1%)
Thunderbird	SuperComputer	10,000,000	ws=100	99,998	79,999 (1.1%)	19,999 (0.1%)
			ws=200	49,998	39,999 (1.6%)	9,999 (0.3%)

5.2.3 Parameter Settings

In our experiments, we employ the default values of hyperparameters that have been widely used in related work [11, 20, 226]. Specifically, we set the layer of the transformer encoder to be 1, the number of attention heads to be 12, and the size of the feed-forward network followed by multi-head self-attention to be 2048. The classification model is optimized by using an improved regularization technique in the Adam optimizer [227], which involves decoupling the weight decay from the gradient-based update. This model was trained with the learning rate of $3e - 4$, the dropout rate is 64, the adopted loss function is weighted binary cross-entropy, and the number of epochs is 20. Furthermore, we use PCA for dimensionality reduction so that the number of components for our self-adaptive clustering is 100, the same as PLELog [13].

To make a fair comparison, we adopt the parameter values provided by the original authors. DeepLog [8] and LogAnomaly [9] adopt a two-layer LSTM with 128 neurons. LogRobust [10] utilizes two BLSTM layers with 128 neurons and one attention layer. PLELog [13] employs a single-layer GRU network for prediction, and its clustering parameter settings remain the same as they introduced (that is, the minimum cluster size and the minimum number of samples are 100) to calculate the number of clusters through HDBSCAN. NeuralLog [11] has the same parameter settings as the transformer encoder we described above. In the original work [9], LogAnomaly applied a synonym- and antonym-based approach to representing log templates as semantic vectors. However, the employed template2vec model was trained with domain-specific antonyms and synonyms, some of which were manually added by operators. Furthermore, the model was trained with the index of log events (i.e., sequential and quantitative vectors), thereby ignoring the semantics of the logs. Due to the unavailability of the information, we follow the empirical study [12], using a pre-trained FastText word2vec model [23] to compute semantic vectors for log templates. The semantic vectors obtained from language models are usually more informative.

The log sequences used in our study are continuous and sequential because of their chronological ordering and grouping based on window size or session window. Hence, the conventional k-fold cross-validation method is not applicable in our timing-related problems, considering the time dependencies present in the data. To assess the effectiveness of AdaLog, we perform each experiment 20 times and report the median value in Table 5.4. These results are more reliable and applicable in practical scenarios.

5.2.4 Evaluation Metrics

Log-based anomaly detection is an imbalanced binary classification problem. To evaluate the effectiveness of AdaLog, we refer to previous works and adopt the $\text{precision} = \frac{TP}{TP+FP}$, $\text{recall} = \frac{TP}{TP+FN}$, and $\text{F1-score} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$ metrics for comparison. In addition, Le et al.'s work [12] demonstrated that $\text{specificity} = \frac{TN}{TN+FP}$ is necessary for evaluation, especially under the imbalanced data distribution. Thus, we introduce this metric in our study. Specifically, TP, TN, FP, and FN refer to the number of true positives (an abnormal log sequence is correctly predicted to be anomalous), true negatives (a normal log sequence is correctly predicted to be normal), false positives (a normal log sequence is incorrectly predicted to be abnormal), and false negatives (an abnormal log sequence is incorrectly predicted to be normal), respectively. Precision indicates the percentage of correctly detected anomalies among all detected anomalies; recall refers to the percentage of correctly detected anomalies over all anomalous log sequences; specificity represents the percentage of correctly detected normal ones over all real normal log sequences; and the F1-score considers both precision and recall, which is a comprehensive evaluation metric. The higher the F1-score value, the more accurate the proposed approach is in predicting both categories (i.e., normal and abnormal).

5.3 Results and Analysis

In Section 5.3.1, we present a comprehensive evaluation of the performance of our proposed AdaLog approach as well as SOTA approaches. Subsequently, we conduct ablation studies to assess the effectiveness of the self-adaptive clustering method, undersampling technique, and overall methodology in Sections 5.3.2, 5.3.3, and 5.3.4, respectively. In Section 5.3.2, we compare our proposed clustering method against the HDBSCAN clustering method used in PLELog, both without (Table 5.5) and with (Table 5.6) the classification model. Additionally, we investigate the efficacy of the undersampling method at different sampling ratios in Section 5.3.3. Finally, in Section 5.3.4, we analyze how varying sizes of labeled data impact the performance of AdaLog. These sections provide a detailed analysis of our experimental results, further contributing to the overall efficacy and understanding of the proposed approach.

In Table 5.4–5.8, the column M indicates the evaluation metrics, and P, R, S, and F1 represent precision, recall, specificity, and F1-score, respectively. For each dataset,

the grouping methods are abbreviated as *session* or the values of the *window sizes* (i.e., 20, 100, and 200).

Table 5.4 Performance of different approaches on three datasets. LogRobust and NeuralLog are supervised, while the others are semi-supervised.

Model	Metric	HDFS session	BGL			Thunderbird		
			20	100	200	20	100	200
DeepLog	P	0.835	0.128	0.166	0.192	0.004	0.017	0.005
	R	0.994	0.995	0.988	0.987	0.938	0.963	1.000
	S	0.994	0.539	0.530	0.528	0.899	0.922	0.005
	F1	0.908	0.227	0.285	0.322	0.008	0.033	0.010
LogAnomaly	P	0.886	0.136	0.176	0.203	0.004	0.025	0.000
	R	0.893	0.970	0.985	0.985	0.938	0.963	1.000
	S	0.961	0.581	0.562	0.559	0.891	0.950	0.005
	F1	0.966	0.239	0.299	0.336	0.008	0.050	0.009
PLELog	P	0.893	0.592	0.595	0.862	0.429	0.826	0.692
	R	0.979	0.882	0.880	0.844	0.688	0.704	0.360
	S	0.996	0.958	0.968	0.985	1.000	1.000	1.000
	F1	0.934	0.708	0.710	0.853	0.528	0.760	0.474
LogRobust	P	0.961	0.616	0.696	0.684	0.377	0.318	0.289
	R	1.000	0.969	0.968	0.963	0.876	1.000	0.960
	S	0.989	0.959	0.960	0.949	0.999	0.997	0.994
	F1	0.980	0.753	0.810	0.800	0.531	0.482	0.444
NeuralLog	P	0.992	0.976	0.975	0.906	0.977	1.000	0.833
	R	0.979	0.979	0.867	0.791	0.935	0.379	0.200
	S	1.000	0.998	0.985	0.991	1.000	1.000	1.000
	F1	0.986	0.977	0.918	0.845	0.956	0.550	0.323
AdaLog	P	0.988	0.970	0.883	0.875	0.912	1.000	0.818
	R	0.997	0.940	0.961	0.847	0.674	0.621	0.360
	S	1.000	0.998	0.988	0.986	1.000	1.000	1.000
	F1	0.993	0.955	0.920	0.861	0.775	0.766	0.500

5.3.1 Detection Accuracy

As shown in Table 5.4, in terms of F1-score, compared with **all semi-supervised approaches**, our approach AdaLog performs best on all datasets, **improving existing approaches by 2489.8% (DeepLog), 2448.8% (LogAnomaly), and 17.8% (PLELog)** on average. On HDFS dataset, all semi-supervised approaches perform well, and AdaLog outperforms these three approaches by 6.2% on average. On BGL dataset, regardless of window size, DeepLog and LogAnomaly have a relatively low F1-score because of the poor precision; that is, they are likely to mislabel normal log sequences as anomalies. AdaLog is superior to these two approaches by 237.0% and 221.2% on average, respectively. In contrast, PLELog performs better but

still 21.8% worse than AdaLog on average. Every dataset has a class imbalance problem, especially Thunderbird dataset, where abnormal log sequences in the test set only account for 0.1%–0.3% of the entire test set under different window sizes. Beneficial from the undersampling method, AdaLog outperforms other approaches most apparently on this dataset with various window sizes, where AdaLog improves DeepLog, LogAnomaly, and PLELog by 5569.5%, 5491.7%, and 17.7% on average, respectively.

Even though AdaLog is a semi-supervised approach, its performance (F1-score) on all datasets is even **23.8%** and **10.9%** better than the **supervised approaches LogRobust and NeuralLog** on average. On HDFS dataset, AdaLog outperforms LogRobust by 1.3%. On BGL and Thunderbird datasets, AdaLog improves LogRobust by 21.8% and 39.2% for three window sizes, separately. Compared with NeuralLog, even though AdaLog performs worse on BGL and Thunderbird datasets at $ws = 20$, it is on average 31.7% better in other cases and 10.9% better on all datasets. In cases where there is a significant imbalance between the two classes in the training set, AdaLog is found to exhibit suboptimal performance relative to NeuralLog. Adding more anomalous data in the supervised approaches may help alleviate overfitting and improve overall performance.

Finding 1: AdaLog outperforms all semi-supervised approaches and obtains an average improvement of 17.8%-2489.8% in terms of F1-score; AdaLog outperforms supervised approaches in most cases.

Table 5.5 Comparison of the effectiveness of different clustering methods on the training set.

Clustering	Metric	HDFS session	BGL			Thunderbird		
			20	100	200	20	100	200
Self-adaptive	P	0.889	0.815	0.786	0.795	0.049	0.088	0.110
	R	0.973	0.967	0.922	0.893	0.881	0.765	0.717
	S	0.996	0.978	0.971	0.970	0.939	0.915	0.908
	F1	0.930	0.884	0.849	0.841	0.093	0.157	0.190
HDBSCAN	P	0.796	0.784	0.672	0.666	0.015	0.044	0.046
	R	0.898	0.993	0.976	0.892	0.883	0.886	0.583
	S	0.993	0.973	0.944	0.941	0.798	0.795	0.807
	F1	0.844	0.876	0.796	0.762	0.030	0.084	0.085

Table 5.6 Comparison of the effectiveness of different clustering methods on the test set.

Clustering	Metric	HDFS session	BGL			Thunderbird		
			20	100	200	20	100	200
Self-Adaptive	P	0.982	0.942	0.896	0.929	0.900	0.913	0.500
	R	0.994	0.962	0.866	0.800	0.587	0.724	0.240
	S	1.000	0.996	0.991	0.993	1.000	1.000	1.000
	F1	0.988	0.952	0.880	0.859	0.711	0.808	0.324
HDBSCAN	P	0.985	0.848	0.824	0.212	-	0.010	-
	R	0.848	0.960	0.920	0.858	0.000	0.138	0.000
	S	1.000	0.988	0.982	0.642	1.000	0.980	1.000
	F1	0.912	0.901	0.869	0.340	-	0.019	-

5.3.2 Clustering Effectiveness

To compare the clustering algorithms employed in AdaLog and PLELog, we adopt the same methods of preprocessing and generating word embeddings (i.e., BERT). In this experiment, we do not perform undersampling to ensure that differences in comparisons are entirely due to clustering. The results on the training set are demonstrated in Table 5.5, and Table 5.6 indicates the comparison on the test set with the transformer-based classification model for anomaly prediction.

In Table 5.5, we compare our self-adaptive clustering method with HDBSCAN employed in PLELog on the training set. On the whole, our clustering method can more accurately label unlabeled data on all datasets. We can find that both clustering methods perform much better on both HDFS and BGL datasets than Thunderbird dataset. On both datasets, the recall of two clustering methods achieves high values, while the precision calculated by AdaLog is higher than the HDBSCAN adopted by PLELog. On HDFS dataset, our self-adaptive clustering method outperforms HDBSCAN in terms of F1 by 10.2%. The improvements of our clustering method over HDBSCAN by 6.0% on average on BGL dataset. Because of the severe imbalance problem on Thunderbird dataset, two clustering methods have difficulty in correctly classifying log sequences. Overall, on all datasets, our self-adaptive clustering method **outperforms HDBSCAN in the term of F1-score with 64.1% on average**.

In Table 5.5, we directly classify unlabeled data on the training set into normal and abnormal based on label probabilities. Nonetheless, to predict the labels of samples in the test set, we use the Transformer-based classification model for training and validation. Similarly, the results displayed in Table 5.6 also illustrate that our clustering method is far better than HDBSCAN in terms of all evaluation metrics. When combining the HDBSCAN clustering method and the same classification model,

the results are not satisfactory. Even on the Thunderbird dataset, the value of TP is 0 (i.e., no anomaly can be correctly predicted), which results in a recall of 0 and makes precision and F1-score invalid values. The reason is that the label probability calculated by HDBSCAN has a large error. Incorporating the classification model, the self-adaptive clustering **outperforms HDBSCAN by 864.1%** (on average) in terms of **F1-score** on three datasets.

Finding 2: Our self-adaptive clustering method outperforms HDBSCAN by 64.1% (training set) and 864.1% (test set) in terms of F1-score.

5.3.3 Undersampling Effectiveness with Different Ratios

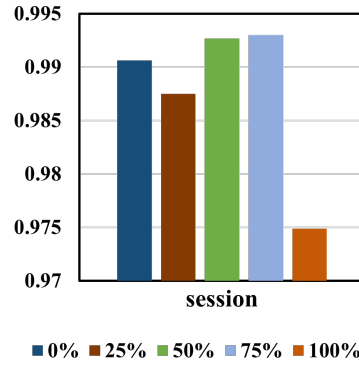
In Table 5.7, we summarize the raw ratios of normal and abnormal data, the undersampling ratio ranges, and the specific undersampling ratios used in AdaLog (i.e., 75% of the range as it is practice-recommended on all three datasets). Table 5.8 shows the comparison of AdaLog with (75% of the range) or without undersampling. For the sake of rigor, we choose five values within the undersampling ratio ranges to observe the effect of the undersampling ratio on AdaLog’s performance. The trend is illustrated in Figure 5.2.

Table 5.7 Statistics on the undersampling ratio of datasets.

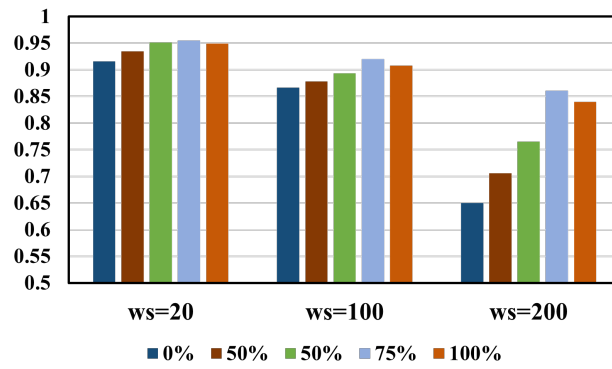
Dataset	Group	Original Ratio	Undersampling Ratio Range	Ratio (75%)
HDFS	session	33.5:1	15:1-30:1	26:1
	20	10.0:1	5:1-9:1	8:1
BGL	100	8.5:1	4:1-8:1	7:1
	200	7.6:1	3:1-7:1	6:1
Thunderbird	20	282:1	140:1-280:1	245:1
	100	93:1	45:1-90:1	79:1
	200	63:1	30:1-60:1	53:1

The results shown in Table 5.8 demonstrate that the performance with undersampling outperforms without undersampling except for only one case (i.e., on Thunderbird dataset with $ws = 100$). The possible reason is that the percentage of anomalies in the training set is much larger than that in the test set, and the use of undersampling may enlarge the difference between the two and lead to a little deviation in prediction. Overall, compared with the model without undersampling, the one with undersampling improves the performance on average by 0.5% (HDFS), 1.7% (BGL), and 19.4% (Thunderbird), respectively.

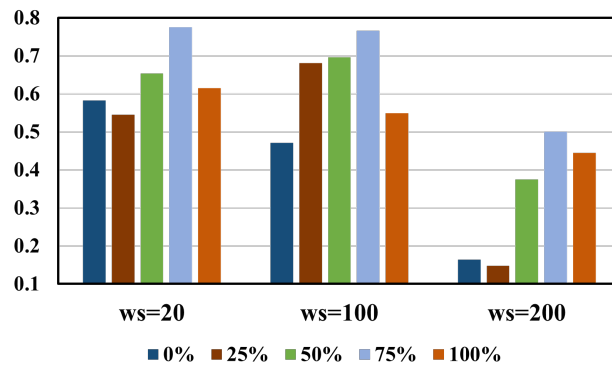
Chapter 5. AdaLog: A Semi-supervised Approach for Industrial Anomaly Detection via Self-adaptive Clustering



(a) HDFS



(b) BGL



(c) Thunderbird

Fig. 5.2 The effect of the undersampling ratio on AdaLog's performance in F1-score. The percentages 0% and 100% indicate the minimum and maximum values of the undersampling range.

Table 5.8 The results of AdaLog without or with undersampling.

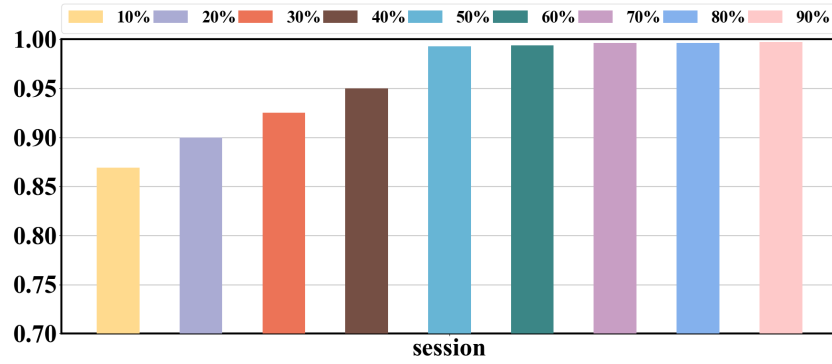
AdaLog	Metric	HDFS session	BGL			Thunderbird		
			20	100	200	20	100	200
W/o	P	0.982	0.942	0.896	0.929	0.900	0.913	0.500
	R	0.994	0.962	0.866	0.800	0.587	0.724	0.240
	S	1.000	0.996	0.991	0.993	1.000	1.000	1.000
	F1	0.988	0.952	0.880	0.859	0.711	0.808	0.324
W/h	P	0.988	0.970	0.883	0.875	0.912	1.000	0.818
	R	0.997	0.940	0.961	0.847	0.674	0.621	0.360
	S	1.000	0.998	0.988	0.986	1.000	1.000	1.000
	F1	0.993	0.955	0.920	0.861	0.775	0.766	0.500

To explore whether this chosen undersampling ratio is reasonable for each dataset, we conduct experiments on the undersampling ratio within a set range shown in Table 5.8. We take different undersampling values, i.e., 0% (minimum), 25%, 50%, 75%, and 100% (maximum), as the undersampling ratio. The performance trend is illustrated in Figure 5.2. The horizontal and vertical axis refers to the percentages of the ratio range and the performance in terms of F1-score, respectively. On the HDFS dataset, performance drops slowly from 0.991 (min) to 0.992 (25%), and rises to 0.993 (75%), then drops sharply to 0.975 at a 30:1 ratio of the two classes (max). Unlike HDFS, the performance of the BGL dataset shows a steady upward trend until reaching 75% of the ratio range. The up and down trends are flatter when the window size is 20 or 100, and the corresponding ranges of the variation are 0.033 and 0.054, respectively. In comparison, the change (i.e., 0.211) is more significant in the case of $ws = 200$. The performance climbed from 0.650 (min) to 0.765 (50%), followed by a sharp rise to 0.861 (75%), and then a slow decline to 0.840 (max).

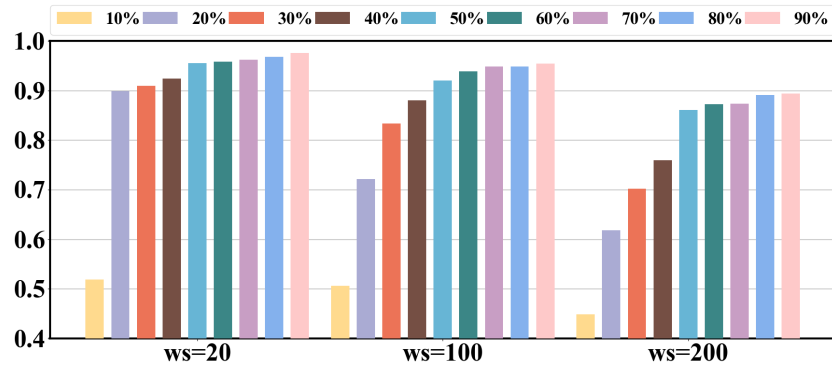
On Thunderbird, the most imbalanced dataset, the variation range widens to 0.352 ($ws = 200$). When the window size is 20 or 200, the trend of change is similar. There is a modest decrease and then a considerable increase, followed by a minor decay. At the peak points, the percentage of the ratio range is 75%. When $ws = 100$, the performance continues to rise to 0.766 as the percentage increases from min to 75%, after which the performance decreases to 0.549 (max). To sum up, for any dataset, within the range of undersampling ratios, 75% is a very reasonable undersampling ratio.

Finding 3: Adopting undersampling is able to improve AdaLog performance. 75% is a reasonable undersampling ratio for various datasets.

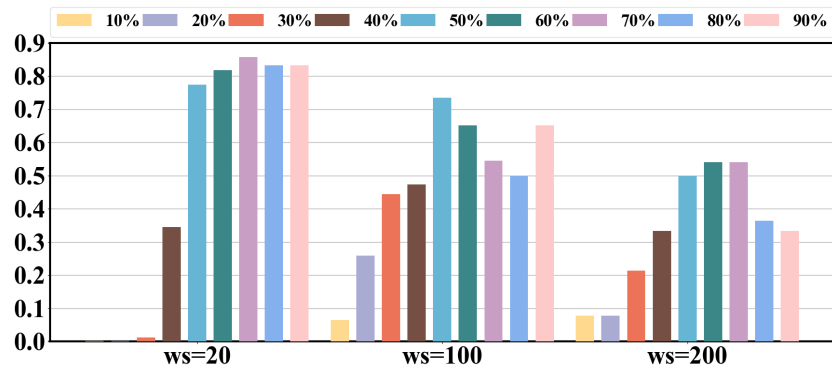
Chapter 5. AdaLog: A Semi-supervised Approach for Industrial Anomaly Detection via Self-adaptive Clustering



(a) HDFS



(b) BGL



(c) Thunderbird

Fig. 5.3 The effect of the labeled training data size on AdaLog's performance in F1-score.

5.3.4 Overall Effectiveness with Different Labeled Ratios

As a semi-supervised approach, AdaLog utilizes a part of labeled training data to predict the unknown ones. The operation benefits the model will be less sensitive to the data, thereby keeping stability. Figure 5.3 represents how the performance of the proposed AdaLog in F1-score changes when the labeled proportion (represented as lp) ranges (i.e., 10% to 90% in 10 percentages).

On HDFS dataset, the performance of AdaLog continues to rise steadily, from 0.869 to 0.997. Especially when lp increases from 50% to 90%, the performance gap of AdaLog is tiny, and the improvement is only 0.4%. This result represents that AdaLog on HDFS dataset maintains outstanding performance even with the proportion of labeled data changes. On BGL dataset, the changing trend of AdaLog's performance is relatively consistent in different window sizes. The performance change of AdaLog under different window sizes is basically a sharp improvement first (especially the lp changes from 10% to 20%) and then a slow increase (i.e., 50%–90%). When $ws = 20$, the F1-score dramatically increases from 0.519 to 0.899 (an improvement of 73.2%) as lp changes from 10% to 20%, then gradually grows to 0.955 when lp is 50%. Similarly, under $ws = 100$, the F1-score has an improvement of 42.5% with the changes of lp from 10% to 20%. Afterward, this value rises by 27.6% until lp increases to 50%. When $ws = 200$, the F1-score greatly improves from 0.449 to 0.861 (an improvement of 91.8%) with the change in lp from 10% to 50%. When the labeled ratio changes from 50% to 90%, the performance of AdaLog is slightly improved by 2.2% ($ws = 20$), 3.7% ($ws = 100$), and 3.8% ($ws = 200$), respectively. The experimental results demonstrate that as the amount of labeled data increases, the performance of AdaLog on BGL dataset also steadily improves. However, when the proportion of labeled data changes from 50% to 90%, the performance does not change much.

AdaLog has an unusual behavior on the Thunderbird dataset compared to the other two datasets. In Table 5.4, supervised methods generally perform better than semi-supervised methods since the former use more semantic information for training. Then on the Thunderbird dataset, due to the severe imbalanced problem, supervised methods will likely overfit anomalies. Once some anomalies that the model has not seen appear, the model is likely to predict them as normal logs. Figure 5.3(c) clarifies this point intuitively. When the window size is 20, the performance of AdaLog spikes from 0.001 ($lp = 10\%$) to 0.775 ($lp = 50\%$), then slowly rises to 0.857 ($lp = 70\%$) before dropping slightly to 0.833 ($lp = 90\%$). When the window sizes are 100 and 200, the performances of AdaLog change significantly. They rise sharply to

Chapter 5. AdaLog: A Semi-supervised Approach for Industrial Anomaly Detection via Self-adaptive Clustering

0.735 ($lp = 50\%$) and 0.541 ($lp = 60\%$), respectively, then drop magnificently. The experimental findings suggest that augmenting the amount of training data is not invariably conducive to enhanced performance, and the data category should be considered. In instances of pronounced class imbalance, such a practice may engender training drift, yielding diminished recall and F1-score. In summary, AdaLog performs well as a semi-supervised method with 50% labeled training data, even better with extreme imbalance.

Finding 4: As the labeled data increases, AdaLog performance normally increases sharply and then steadily. On datasets with severe class imbalance, augmenting the labeled data does not always lead to better performance.

5.4 Chapter Summary

In this chapter, we have proposed an efficient semi-supervised method AdaLog to enhance the performance of anomaly detection. To improve the accuracy of data labeling, AdaLog utilizes a self-adaptive clustering method to calculate the label probability of unlabeled data more accurately, which considers twelve specially-designed situations based on the distance of labeled data. To alleviate the class imbalance problem, AdaLog employs an undersampling method to improve the performance of AdaLog. Furthermore, we adopt a pre-trained model for word embedding and a Transformer-based model for prediction. Experimental results show that AdaLog achieves superior performance than five SOTA semi-supervised and supervised approaches for industrial log-based anomaly detection.

Chapter 6

On the Influence of Data Resampling for Deep Learning-Based Log Anomaly Detection: Insights and Recommendations

Numerous Deep Learning (DL)-based approaches have garnered considerable attention in the field of software Log Anomaly Detection (LAD). However, a practical challenge persists: The prevalent issue of class imbalance in the public data commonly used to train the DL models. This imbalance is characterized by a substantial disparity in the number of abnormal log sequences compared to normal ones, for example, anomalies represent less than 1% of one of the most popular datasets, namely the Thunderbird dataset. Previous research has indicated that existing DLLAD approaches may exhibit unsatisfactory performance, particularly when confronted with datasets featuring severe class imbalances. Mitigating class imbalance through data resampling has proven effective for other software engineering tasks, however, it has been unexplored for LAD thus far. This study aims to fill this gap by providing an in-depth analysis of the impact of diverse data resampling methods on existing DLLAD approaches from two distinct perspectives. Firstly, we assess the performance of these DLLAD approaches across three datasets, and explore the impact of resampling ratios of normal to abnormal data on ten data resampling methods. Secondly, we evaluate the effectiveness of the data resampling methods when utilizing optimal resampling ratios of normal to abnormal data. Our findings indicate that oversampling methods generally outperform undersampling and hybrid sampling methods. Data resampling

Chapter 6. On the Influence of Data Resampling for Deep Learning-Based Log Anomaly Detection: Insights and Recommendations

on raw data yields superior results compared to data resampling in the feature space. In most cases, certain undersampling and hybrid methods (e.g., SMOTEENN and InstanceHardnessThreshold) show limited effectiveness. Additionally, by exploring the resampling ratio of normal to abnormal data, we suggest generating more data for minority classes through oversampling while removing less data from majority classes through undersampling. In conclusion, our study provides valuable insights into the intricate relationship between data resampling methods and DLLAD. By addressing the challenge of class imbalance, researchers and practitioners can enhance the model performance in DLLAD. This work was submitted to IEEE Transactions on Software Engineering in 2024.

The subsequent sections of this chapter are organized as follows: Section 6.1 introduces the background of DLLAD and data resampling methods. Section 6.2 illustrates the study design, while Section 6.3 analyzes the results and presents findings. Sections 6.4 and 6.5 discuss our study and threats to validity, separately. Finally, Section 6.6 summarizes the chapter's work.

6.1 Background

6.1.1 Overview of DLLAD Models

The typical workflow of DLLAD approaches (shown in Figure 6.1) consists of four phases: 1) log parsing, 2) log grouping, 3) log embedding, and 4) model training and prediction.

To effectively extract valuable information for analysis, previous studies [39, 8, 10, 9, 11] convert the unstructured log messages generated during system operation into structured log events. Each log message comprises a header and content, where the header includes information like timestamps, typically omitted from analysis [11]. The log content is then segmented into constant and variable sections [11]. By replacing variable elements with a special symbol, the original log messages are converted into log events as illustrated in Figure 6.1 To group log events into log sequences, we adopt the fixed window strategy used in prior studies [8, 9, 13]. If an abnormal log event is part of a log sequence, the log sequence is labeled as abnormal. Conversely, if the log sequence comprises solely normal log events, it is labeled as normal. These log sequences are subsequently transformed into embedding vectors and used as input for training a classification model to predict whether a log sequence is abnormal.

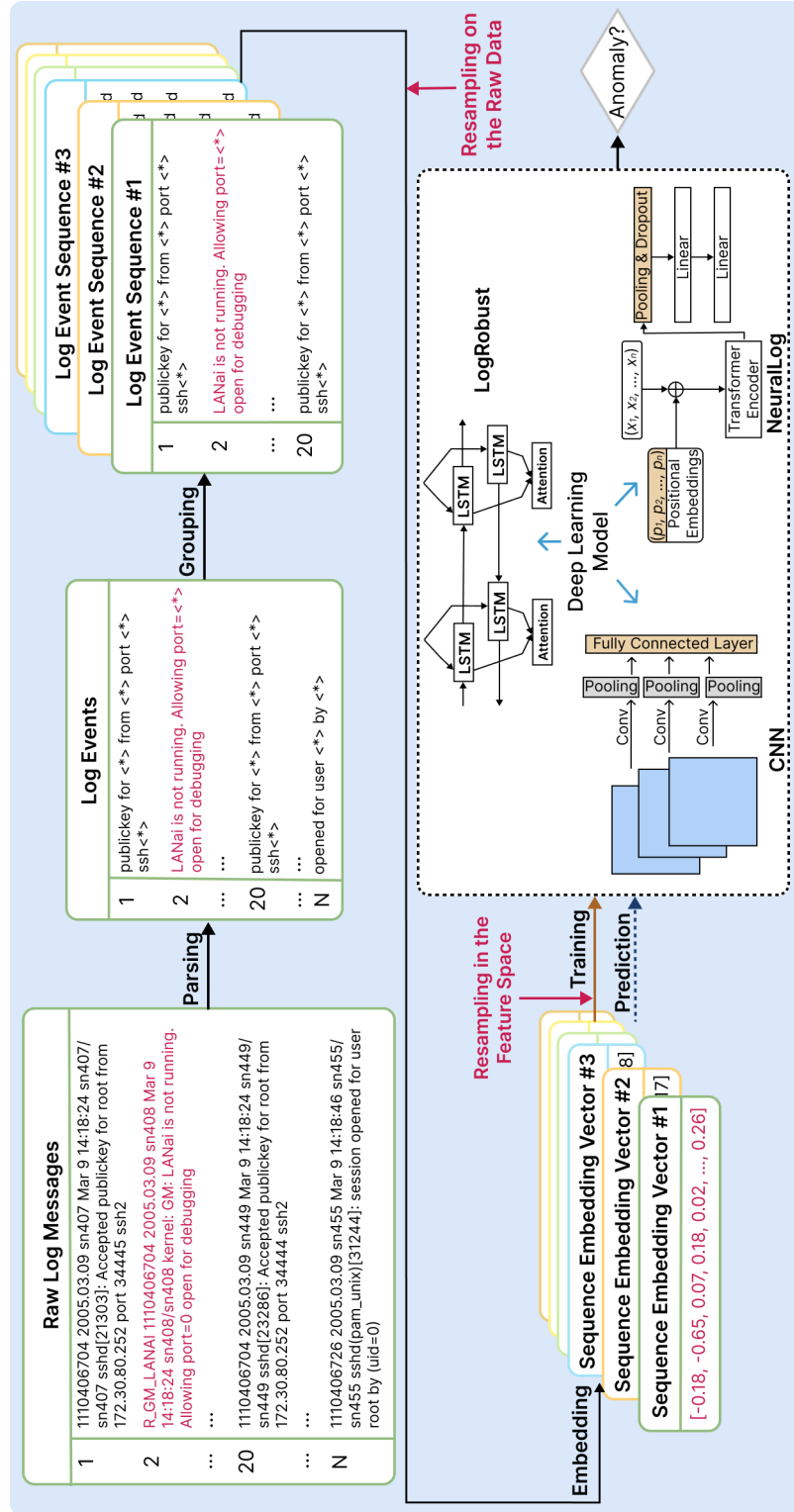


Fig. 6.1 The common workflow of DLLAD approaches.

6.1.2 Existing DLLAD Approaches

Recent DL approaches for anomaly detection can be categorized into three main groups: Convolutional Neural Network (CNN)-based models, Long Short-Term Memory-based models, and Transformer-based models. To select the most suitable DLLAD approaches, we take two factors into consideration. Firstly, we look for models that are representative of the DL models described in Section 2.3.1. Secondly, we aim to include models that have been recently proposed. Therefore, we choose the following models:

CNN. Lu et al. [47] adopted a CNN-based model to automatically detect log anomalies. Logs were parsed into templates based on log keys, which were then encoded using logkey2vec. These embeddings were structured into a trainable matrix, simplifying neural network training. The model architecture comprised three convolutional layers, a dropout layer, and max-pooling layers.

LogRobust. Zhang et al. [10] employed Drain [228] for log parsing and integrated the FastText [23], a pre-trained Word2vec model, with TF-IDF weights [229] to represent log events as semantic vectors. Subsequently, these vectors are utilized as input to an attention-based Bi-directional LSTM (Bi-LSTM) model for detecting anomalies.

NeuralLog. Le et al. [11] preprocessed log messages without log parsing and encoded them into vector representations via a pre-trained Transformer-based language model BERT [20]. Then, they apply a transformer encoder to classify log sequences, with the primary objective of capturing semantic information comprehensively.

6.1.3 Data Resampling

We make use of commonly adopted data resampling methods in the software engineering domain [230, 231, 121, 232]. These methods are classified into three primary categories: oversampling, undersampling, and hybrid sampling. As depicted by two arrows in Figure 6.1, certain resampling methods are tailored for use exclusively within the feature space, while others, such as random oversampling and undersampling [231], can be applied to both the raw data and the feature space. Data resampling methods applied to raw data focus on directly modifying the distribution of log sequences in the training set to address class imbalance, thereby altering the number of log sequences belonging to each class and subsequently impacting the embedding vectors derived from them. In contrast, data resampling methods within the feature

space involve first embedding the original log sequences and then resampling based on these embedding vectors.

(1) Random OverSampling (ROS) randomly replicates log sequences of the minority class without generating new ones. These replicated abnormal sequences are then added to the original dataset. This method, when applied to raw data and feature space, is denoted as ROS_R and ROS_F , respectively.

(2) Random UnderSampling (RUS) randomly selects log sequences from the majority class and subsequently removes them from the original dataset. This method, when applied to raw data and feature space, is denoted as RUS_R and RUS_F , respectively.

(3) Synthetic Minority Oversampling Technique (SMOTE) [49] is an oversampling method applied to the feature space. It augments the minority class by generating synthetic log sequences instead of duplications. This method first randomly selects log sequences from the minority class. For each selected abnormal log sequence M_A , one of its k nearest neighbors M_B are randomly chosen. The embedding vector of the synthetic log sequence M_S is calculated with the formula $x_s = x_A + \text{Random}(0, 1)(x_B - x_A)$, where x_A and x_B represent the embedding vectors of M_A and M_B , separately. These newly generated synthetic abnormal log sequences are subsequently added to the original dataset.

(4) Adaptive Synthetic Sampling Approach [50] (ADASYN) serves as an extension of SMOTE. Unlike SMOTE, ADASYN generates new synthetic abnormal log sequences near the class boundary instead of within the abnormal log sequences themselves.

(5) NearMiss [51] operates as an undersampling method. It calculates the distance between two classes and randomly removes normal log sequences based on the distance. In our evaluation, we adopt NearMiss-3, which has demonstrated superior performance compared to NearMiss-1 and NearMiss-2. Specifically, NearMiss-3 selects a number of the nearest normal log sequences for each abnormal log sequence and removes them from the dataset.

(6) Instance Hardness Threshold (IHT) [52] involves the application of a classifier to the dataset, followed by the removal of log sequences that are hard to classify. The Random Forest (RF) [180] algorithm serves as the default estimator for estimating the Instance Hardness (IH) [52] of individual log sequences.

(7) SMOTEENN [53] is a hybrid sampling method that combines the oversampling method SMOTE and the undersampling method Editd Nearest Neighbour (ENN) [233]. SMOTE generates abnormal log sequences that can sometimes overlap with

the majority class, making classification challenging. ENN, acting as a data cleaning method, helps address this issue. For a normal log sequence, if more than half of its k neighbor log sequences do not belong to the majority class, then the log sequence is removed.

(8) SMOTETomek [53] is a hybrid sampling method that shares similarities with SMOTEENN. It incorporates Tomek links [234] for data cleaning, defined by the distances between log sequences M_i and M_j from different two classes. A pair (M_i, M_j) forms a Tomek link if no log sequence M exists with $d(M_i, M) < d(M_i, M_j)$ or $d(M_j, M) < d(M_i, M_j)$. After oversampling by SMOTE, the log sequences that form Tomek links are then removed to help reduce potential noise or borderline log sequences that may affect classification performance.

6.2 Study Design

6.2.1 Datasets

To assess the performance of the log anomaly detection approaches with the data resampling methods, there are four widely-used publicly available datasets [42, 235], namely HDFS, BGL, Thunderbird, and Spirit. However, since most existing approaches (e.g., LogRobust and NeuralLog) have already achieved satisfactory results on the HDFS dataset (e.g., F1 exceeding 0.98), we exclude it from our evaluation. BGL dataset [42] is a collection of supercomputing system log data gathered by Lawrence Livermore National Labs. Thunderbird and Spirit datasets [42] are acquired from two real-world supercomputers at Sandia National Labs. These log datasets consist of both normal and abnormal log messages, which have been manually identified.

To group log events into a log sequence, a fixed window grouping strategy is commonly used [8, 9, 13]. However, choosing an appropriate window size (ws) is challenging. A small ws makes it difficult for log anomaly detection models to capture anomalies that span multiple log sequences [12]. Additionally, employing smaller ws results in more log sequences containing fewer log events, ultimately leading to slower training speed. On the other hand, if ws is large, log sequences may include multiple anomalies and confuse the detection scheme [12, 236]. In the majority of prior research studies [8, 9, 11, 13, 115], a single window size is typically employed to evaluate the proposed approaches, with $ws=20$ being the most common choice. A few studies [12, 48] have investigated multiple window sizes including 20, 100, and 200. In most cases, the F1 performance is found to be better at $ws=20$ and 100 compared to $ws=200$.

Table 6.1 Statistics of the three public datasets. TB, *Msg*, *Seq*, and *A* are the abbreviations of Thunderbird, Messages, Sequences, and Anomalies, respectively.

Dataset	# of <i>Msg</i>	<i>ws</i>	Training Data		Testing Data	
			# of <i>Seq</i>	# of <i>A</i>	# of <i>Seq</i>	# of <i>A</i>
BGL	4,713,493	20	188,540	17,252	47,134	3,006
		50	75,416	7,415	18,853	1,383
		100	37,708	4,009	9,425	817
TB	5,000,000	20	200,000	328	50,000	37
		50	79,999	195	19,999	29
		100	39,999	138	9,999	23
Spirit	5,000,000	20	200,000	8,817	50,000	290
		50	79,999	4,275	19,999	270
		100	39,999	2,577	9,999	250

However, there is no consistent pattern regarding which performs better between $ws=20$ and $ws=100$. Our experiment results (shown in Table 6.3) also emphasize the absence of a universally optimal window size across all DLLAD datasets. For instance, LogRobust exhibits superior F1 and MCC performance on the Thunderbird dataset at $ws=100$, while achieving better performance on other datasets at $ws=20$. As a result, in our experiments, we consider both $ws=20$ and $ws=100$ as window sizes to account for potential variations in performance. Additionally, we introduce a ws of 50 to provide a balanced perspective between the shorter and longer sequences analyzed. By including this intermediate window size, we aim to uncover a more nuanced understanding of how log sequence length impacts DLLAD performance, and whether the effects of data resampling across datasets with different window sizes are robust.

In Table 6.1, we present the number of log sequences (# of Sequences) in each dataset across various window sizes, as well as recording the number of abnormal sequences within all sequences (# of Anomalies) in both the training and test sets. In Table 6.2, we report the proportions of abnormal sequences within all sequences (% of Anomalies) before and after employing data resampling methods. The proportions of anomalies are determined based on the resampling ratios of normal to abnormal log sequences, which are calculated by multiplying the original ratio of normal to abnormal log sequences by quarter-based constants. The original datasets exhibit very small proportions of anomalies, ranging from 0.16% to 10.63%. Moreover, enlarging the window size has minimal impact on the level of class imbalance across each dataset (for example, in BGL, the proportion of abnormal sequences is 9.15% with $ws=20$ and 10.63% with $ws=100$). Upon implementing the specified resampling ratios, the

Chapter 6. On the Influence of Data Resampling for Deep Learning-Based Log Anomaly Detection: Insights and Recommendations

proportions of abnormal sequences have shown increases: 9.15% to 40.29% (BGL dataset with $ws=20$), 9.83% to 43.62% (BGL dataset with $ws=50$), 10.63% to 49.33% (BGL dataset with $ws=100$), 0.16% to 0.66% (Thunderbird dataset with $ws=20$), 0.24% to 0.98% (Thunderbird dataset with $ws=50$), 0.35% to 1.38% (Thunderbird dataset with $ws=100$), 4.41% to 18.50% (Spirit dataset with $ws=20$), 5.34% to 22.58% (Spirit dataset with $ws=50$), 6.44% to 27.55% (Spirit dataset with $ws=100$), respectively.

6.2.2 Evaluation

We use four commonly used evaluation metrics Recall, Precision, Specificity, and F1-score in previous DLLAD studies [13, 11, 12, 14]. Given that Matthews Correlation Coefficient (MCC) and Area Under the Curve (AUC) are recommended for evaluating software engineering tasks with class imbalance [237–239, 125, 240], we include both MCC and AUC in our evaluation to provide a comprehensive assessment of DLLAD model performance. The commonly used four metrics originate from the confusion matrix, which describes four types of instances: TP (True Positives) represents the number of abnormal log sequences correctly predicted as anomalies, TN (True Negatives) represents the number of normal log sequences correctly predicted as normal, FP (False Positives) represents the number of normal log sequences incorrectly predicted as anomalies, and FN (False Negatives) represents the number of abnormal log sequences incorrectly predicted as normal. The definitions of these metrics are as follows:

(1) $Recall = \frac{TP}{TP+FN}$ represents the proportion of actual anomalies that are correctly predicted by DLLAD models out of all actual anomalies present in the testing dataset. It indicates DLLAD models' ability to capture all abnormal log sequences correctly.

(2) $Precision = \frac{TP}{TP+FP}$ measures the proportion of predicted anomalies by DLLAD models that are actual anomalies out of all anomalies predicted by the models. It indicates the accuracy of the DLLAD models in identifying actual anomalies without falsely labeling normal log sequences as anomalies.

(3) $Specificity = \frac{TN}{TN+FP}$ represents the proportion of actual normal log sequences that are correctly predicted as normal by DLLAD models out of all actual normal log sequences. It indicates the ability of the DLLAD models to correctly identify normal log sequences as normal.

Table 6.2 The proportion of abnormal sequences within all log sequences before and after over-/under-/hybrid sampling. r denotes the original ratio of normal to abnormal log sequences in the training dataset, and quarter-based constants are represented as $1/4$, $1/2$, and $3/4$. The last nine columns correspond to the proportion of abnormal sequences after data resampling.

Dataset	ws	% of Anomalies	% of Anomalies After Oversampling			% of Anomalies After Undersampling			% of Anomalies After hybrid sampling		
			$r^*1/4$	$r^*1/2$	$r^*3/4$	$r^*1/4$	$r^*1/2$	$r^*3/4$	$r^*1/4$	$r^*1/2$	$r^*3/4$
BGL	20	9.15%	40.29%	20.14%	13.43%	28.72%	16.77%	11.84%	38.84%	19.36%	12.90%
	50	9.83%	43.62%	21.81%	14.54%	30.37%	17.90%	12.69%	43.27%	21.62%	14.39%
	100	10.63%	47.59%	23.79%	15.86%	32.24%	19.22%	13.69%	49.33%	21.98%	13.53%
TB	20	0.16%	0.66%	0.33%	0.22%	0.65%	0.33%	0.22%	0.58%	0.29%	0.20%
	50	0.24%	0.98%	0.49%	0.33%	0.97%	0.49%	0.32%	0.85%	0.43%	0.32%
	100	0.35%	1.38%	0.69%	0.46%	1.37%	0.69%	0.46%	1.36%	0.67%	0.46%
Spirit	20	4.41%	18.45%	9.22%	6.15%	15.57%	8.44%	5.79%	18.50%	9.20%	6.17%
	50	5.34%	22.58%	11.29%	7.53%	18.42%	10.15%	7.00%	17.70%	10.50%	7.81%
	100	6.44%	27.55%	13.77%	9.18%	21.60%	12.11%	8.41%	22.44%	11.16%	7.39%

Chapter 6. On the Influence of Data Resampling for Deep Learning-Based Log Anomaly Detection: Insights and Recommendations

(4) $F1\text{-score} = \frac{2 \times (\text{Recall} \times \text{Precision})}{\text{Recall} + \text{Precision}}$ calculates the harmonic mean of Recall and Precision. It provides a balanced measure between Precision and Recall, giving equal weight to false positives and false negatives.

(5) $MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$ is a fully symmetric metric that takes into account all four values (TP, TN, FP, and FN) in the confusion matrix when calculating the correlation between ground truth and predicted values.

(6) AUC is a threshold-independent measure that can be calculated by assessing the area under the Receiver Operating Characteristic (ROC) curve, which plots the true positive rate (Sensitivity) against the false positive rate (1 - Specificity) at various threshold settings. Unlike other metrics such as Precision, Recall, F1-score, and MCC, which depend on the choice of a threshold, AUC evaluates the classifier's performance across all possible threshold values.

To determine the statistical significance of the observed performance differences among these data resampling methods, we employ the Scott-Knott Effect Size Difference (ESD) test [241] based on the assumptions of ANalysis Of VAriance (ANOVA). The Scott-Knott ESD test is a multiple comparison approach that leverages hierarchical clustering to partition these data resampling methods into distinct groups, exhibiting statistically significant differences at the predetermined significance level of 0.05 ($\alpha=0.05$). There are no statistically significant differences between data resampling methods within the same group, but significant differences are observed between data resampling methods located in different groups.

6.2.3 Research Questions

RQ1. Do the existing DLLAD approaches perform well enough with varying degrees of class imbalance? In this RQ, we evaluate the performance of existing DLLAD approaches in detecting log sequence anomalies across datasets with different levels of class imbalance. Specifically, our objective is to understand whether significant differences in class imbalance have a notable influence on the effectiveness of these DLLAD approaches. In cases where the achieved performance is unsatisfactory, especially when dealing with extreme class imbalance, the utilization of data resampling methods becomes essential to address data imbalance issues prior to model training. Consequently, in RQ2, our primary focus is to explore the optimal resampling ratios of normal to abnormal data. Subsequently, based on these ratios, in RQ3, we discuss the effectiveness of data resampling methods in enhancing the capabilities of DLLAD models.

RQ2. How does the resampling ratio of normal to abnormal data affect existing DLLAD model performance? Due to the varying levels of class imbalance in the different datasets, it is challenging to establish a fixed resampling ratio of normal to abnormal data, such as maintaining a consistent 10:1 ratio of normal to abnormal log sequences across all datasets. Conducting an exhaustive exploration of countless potential ratios to identify an optimal resampling ratio for each dataset is not practically feasible. Considering the substantial data imbalance, we avoid pursuing a 1:1 ratio of normal to abnormal data during the data resampling process. This is particularly evident in the Thunderbird dataset, where the training set has a mere 0.16% anomaly rate with a window size of 20. Applying a 1:1 ratio in such cases would result in excessive data removal during undersampling, leading to a significant loss of information. Thus, this resampling ratio is not considered. Instead, we adopt the quarter as a foundational unit for our empirical investigations in a flexible and adaptive manner, as shown in Table 6.2.

RQ3. Does data resampling improve the effectiveness of existing DLLAD approaches? In this RQ, we utilize the recommended resampling ratios of normal to abnormal data observed in RQ2 within evaluated data resampling methods. Subsequently, we assess the effectiveness of these resampling methods when applied to existing DLLAD approaches.

6.2.4 Implementations

We implement the existing approaches introduced in Section 6.1.2 using their respective GitHub repositories or reproduced codebases. Following previous works [11, 12], in our dataset setup, the training set comprises the first 80% of raw logs, while the remaining 20% is allocated for testing. To reduce computational complexity and memory demands during data resampling operations, for NeuralLog [11], we reduce the embedding dimension from 768 to 256. For CNN [47] and LogRobust [10], we utilize the implementations [12], adhering to the instructions provided by the authors. The implementation of data resampling methods is carried out using the Python toolbox [242] Imbalanced-learn¹. Our experiments encompass 27 distinct instances, resulting from the combination of 3 DLLAD approaches, 3 datasets, and 3 window sizes. For each experimental instance, we investigate 10 data resampling methods with 3 different resampling ratios of normal to abnormal data and *NoSampling*. As a result, we have a total of $3 \times 3 \times 3 \times (10 \times 3 + 1)$ combinations, summing up to 837 unique scenarios.

¹<https://imbalanced-learn.org/stable/references/index.html>

To mitigate the variations in performance across different runs, we perform five runs for each data resampling method (including *NoSampling*), culminating in a total of 4,185 experiments conducted in this study. These five-run results are utilized for statistical significance analysis using the Scott-Knott ESD test, which calculates the group ranking of each data resampling method across different datasets. Furthermore, the averages of the five-run results are provided in Tables 6.3–6.6 in Section 6.3. We run our experiments on a Linux server with an Intel Xeon Silver 4210 CPU and four Nvidia GeForce RTX 3090-Ti GPUs.

6.3 Results and Analysis

6.3.1 RQ1. Do the existing DLLAD approaches perform well enough with varying degrees of class imbalance?

Table 6.3 provides an overview of the performance of CNN, LogRobust, and NeuralLog on the BGL, Thunderbird, and Spirit datasets with three window sizes. On the BGL and Thunderbird datasets, the best values for all evaluation metrics are attained with a window size of 20 or 50. Specifically, on the BGL dataset, except for NeuralLog achieving the highest Precision and Specificity with $ws=50$, setting $ws=20$ yields the best values for all evaluation metrics. Regarding the Thunderbird dataset, LogRobust exhibits notably low precision with $ws=20$, resulting in subpar performance in terms of F1 and MCC. However, LogRobust demonstrates better performance with $ws=50$ than $ws=100$ in terms of all evaluation metrics. CNN and NeuralLog follow a similar pattern, with all evaluation metrics achieving optimal results when $ws=20$, except for NeuralLog achieving the highest AUC with $ws=50$. On the Spirit dataset, CNN achieves similar performance with $ws=20$ and $ws=100$ in terms of F1 and MCC, outperforming $ws=50$. LogRobust’s best performance is observed with $ws=20$ in terms of Precision, Specificity, F1, and MCC, while LogRobust performs best in Recall and AUC with $ws=100$. NeuralLog demonstrates comparable performance with $ws=20$ and 50, with slight variations < 0.001 in Specificity, F1, and MCC, but superior to $ws=100$.

Regarding the Thunderbird dataset, it is notable that all three approaches demonstrate relatively poor performance, particularly CNN and LogRobust, which exhibit F1 and MCC values below 0.5. Although NeuralLog performs better with an F1 of 0.760 at $ws=20$, its performance remains unsatisfactory. This subpar performance may be attributed to the dataset’s extreme imbalance across three window sizes,

compromising the models' ability to accurately detect both normal and abnormal log sequences.

Finding 1: When the log sequence contains fewer logs, DLLAD approaches tend to perform better since abnormal logs are more easily noticed in shorter sequences.

Finding 2: When there is a severe data imbalance (e.g., Thunderbird dataset), DLLAD approaches tend to perform poorly because a significant class imbalance can lead to the models becoming biased.

6.3.2 RQ2. How does the resampling ratio of normal to abnormal data affect existing DLLAD model performance?

We comprehensively evaluate the ten data resampling methods for each dataset using quarter-based resampling ratios obtained by multiplying the original ratio of normal data to abnormal data by $1/4$, $1/2$, and $3/4$, as described in Section 6.2.1. The employed data resampling methods are categorized into three distinct groups: OverSampling (comprising *SMOTE*, *ADASYN*, *ROS_F*, and *ROS_R*), UnderSampling (encompassing *NearMiss*, *IHT*, *RUS_F*, and *RUS_R*), and HybridSampling, represented by *SMOTEENN* and *SMOTETomek*. In Figure 6.2a, the three DLLAD approaches utilizing the ten data resampling methods with the three resampling ratios are described through heatmaps, with a special focus on the MCC metric, since MCC is a fully symmetric metric that takes into account all four values (TP, TN, FP, and FN) in the confusion matrix. We adopt an enumeration of “hits”, signifying instances where a particular resampling ratio maximizes the performance of these DLLAD approaches.

In Figures 6.2a–6.2c, the analytical scope centers on three datasets that use the same window size of 20. As an illustrative example, the MCC values for NeuralLog's performance using *SMOTE* with three resampling ratios on the BGL dataset ($ws=20$) are 0.947, 0.939, and 0.937, respectively. In this specific case, the resampling ratio equal to one-quarter of the original data ratio of normal to abnormal data yields the highest MCC value. Consequently, a “hit” is recorded for the corresponding entry, aligning with the cell marked with a count of 8 in the lower left corner. If all three MCC values were identical, there would be no hits recorded. In Figure 6.2a, oversampling methods demonstrate superior performance when the resampling ratio is one-quarter of the original ratio of normal to abnormal data. This phenomenon is particularly

Table 6.3 The Recall, Precision, Specificity, F1-score, MCC, and AUC values of the three DLLAD approaches across the three datasets, each with three different window sizes. Bold font highlights the best performance among the three window sizes.

Model	Metric	BGL			TB			Spirit		
		$w_s=20$	$w_s=50$	$w_s=100$	$w_s=20$	$w_s=50$	$w_s=100$	$w_s=20$	$w_s=50$	$w_s=100$
CNN	R	0.948	0.907	0.916	0.495	0.460	0.304	0.815	0.893	0.909
	P	0.985	0.708	0.362	0.443	0.169	0.213	0.797	0.582	0.726
	S	0.999	0.965	0.827	0.999	0.997	0.997	0.999	0.991	0.991
	F1	0.966	0.787	0.510	0.441	0.247	0.241	0.806	0.704	0.807
	MCC	0.964	0.779	0.509	0.454	0.277	0.247	0.805	0.716	0.807
	AUC	0.977	0.919	0.903	0.866	0.812	0.786	0.967	0.996	0.995
LogRobust	R	0.949	0.911	0.903	0.766	0.609	0.449	0.892	0.910	0.918
	P	0.860	0.709	0.793	0.084	0.317	0.277	0.973	0.805	0.863
	S	0.989	0.968	0.978	0.994	0.998	0.997	1.000	0.997	0.994
	F1	0.903	0.792	0.844	0.151	0.415	0.340	0.930	0.832	0.889
	MCC	0.897	0.783	0.831	0.252	0.437	0.350	0.931	0.854	0.885
	AUC	0.970	0.960	0.954	0.915	0.861	0.826	0.991	0.990	0.997
NeuralLog	R	0.896	0.627	0.598	0.772	0.730	0.756	0.899	0.931	0.938
	P	0.852	0.872	0.671	0.758	0.469	0.470	0.899	0.864	0.800
	S	0.989	0.991	0.878	1.000	0.999	0.999	0.999	0.999	0.999
	F1	0.872	0.721	0.496	0.760	0.571	0.579	0.895	0.896	0.862
	MCC	0.864	0.718	0.511	0.762	0.585	0.596	0.897	0.896	0.865
	AUC	0.943	0.809	0.738	0.856	0.865	0.828	0.949	0.965	0.964

6.3 Results and Analysis

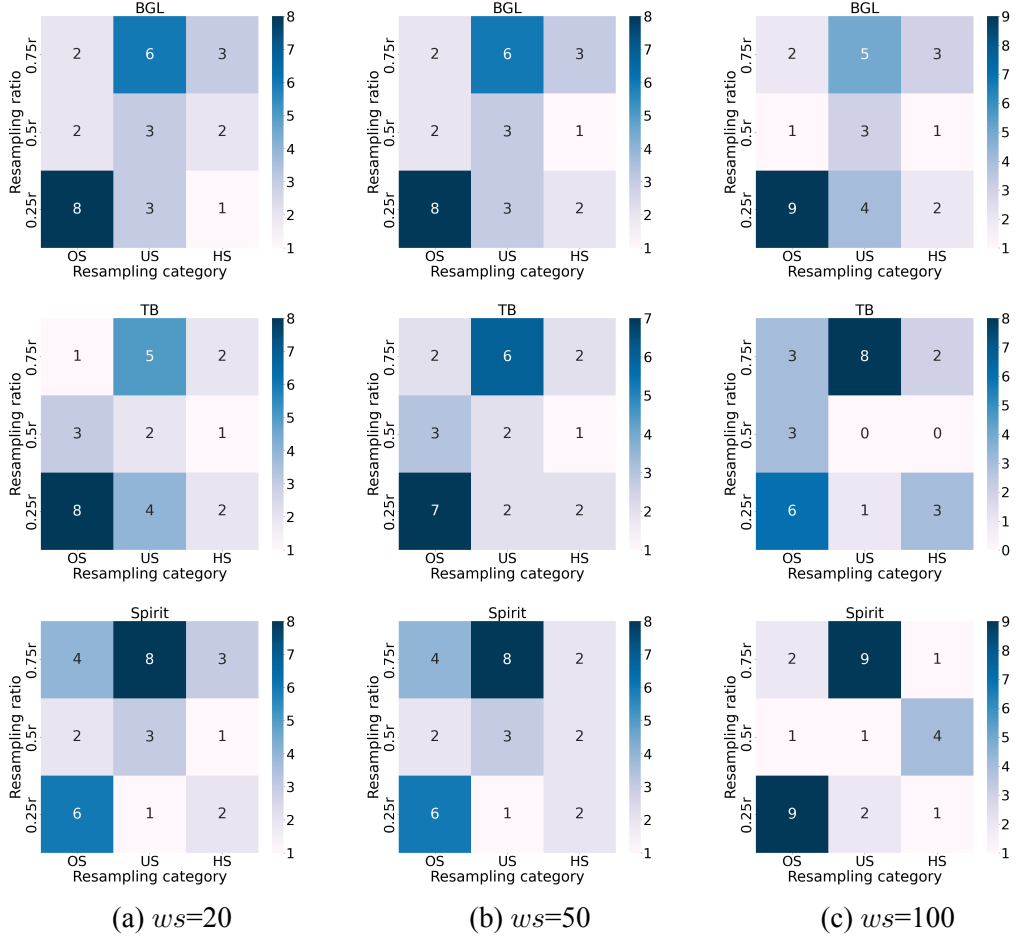


Fig. 6.2 The heatmap depicting the statistics of the best performance of the data resampling methods (including OverSampling (OS), UnderSampling (US), and HybridSampling (HS)) with the three resampling ratios (r representing the original ratio of normal to abnormal data) across the three datasets with different window sizes. (a) $ws=20$. (b) $ws=50$. (c) $ws=100$.

Chapter 6. On the Influence of Data Resampling for Deep Learning-Based Log Anomaly Detection: Insights and Recommendations

pronounced in BGL and Thunderbird datasets, both yielding 8 hits out of a total of 12. Regarding the Spirit dataset, the DLLAD approaches with *NoSampling* exhibit strong performance in accurately detecting anomalies, with the MCC ranging from 0.716 to 0.931. Interestingly, for this dataset, the selected resampling ratio of normal to abnormal data appears to have less impact on the performance of oversampling methods. When considering undersampling methods, removing a smaller amount of original normal log sequences can enhance the effectiveness of these methods. This effect is particularly prominent when examining the Spirit dataset, yielding 8 out of 12 hits. For hybrid sampling methods, it is difficult to determine the most appropriate resampling ratio of normal to abnormal data for the studied approaches. Especially in the Thunderbird dataset, both one-quarter and three-quarter resampling ratios exhibit an equal likelihood of achieving the best DLLAD performance.

In Figures 6.2b and 6.2c, our findings exhibit a remarkable consistency with those presented in Figure 6.2a. When the resampling ratio is set to one-quarter of the original ratio of normal to abnormal data, employing oversampling methods on the three datasets demonstrates the highest likelihood of achieving optimal performance, particularly in the BGL dataset, with 8 or 9 out of 12 hits. Conversely, when the resampling ratio is adjusted to three-quarters of the original ratio of normal to abnormal data, the effectiveness of undersampling methods is maximized. This is notable in the Spirit dataset, where 8 or 9 out of 12 hits are observed, signifying optimal performance. Concerning hybrid sampling methods, offering robust recommendations regarding the optimal resampling ratio of normal to abnormal data remains a challenging task. The effectiveness varies inconsistently across the three datasets. Specifically, the most suitable resampling ratio within the predefined range is three-quarters of the original ratio for the BGL dataset, one-quarter of the original ratio for the Thunderbird dataset, and one-second of the original ratio for the Spirit dataset, respectively.

Finding 3: DLLAD approaches tend to show robust initial performance (i.e., Figure 6.3) on certain datasets (i.e., Spirit), making the chosen resampling ratio of normal to abnormal data for oversampling relatively less sensitive in these cases.

Finding 4: In general, to enhance the performance of DLLAD approaches with *oversampling*, it is recommended to increase the degree of minority class amplification (i.e., generating more abnormal log sequences). Conversely, for DLLAD approaches employing *undersampling*, it is advisable to reduce the degree of majority class reduction (i.e., removing fewer normal log sequences). Notably, there is no specific recommended resampling ratio preference for *hybrid sampling* methods.

The effectiveness of this finding is mainly attributed to two aspects: 1) the introduction of diverse features through extensive oversampling of abnormal log sequences, and 2) the retention of valuable information from the original data while efficiently removing a limited number of normal log sequences. Oversampling and undersampling under these specific conditions contribute to enhancing the classification capabilities of DLLAD approaches.

6.3.3 RQ3. Does data resampling improve the effectiveness of existing DLLAD approaches?

Tables 6.4–6.6 present the results of the three DLLAD approaches with ten data resampling methods and *NoSampling* on the three datasets. We utilize three distinct colors to underscore the statistical differences among data resampling methods and *NoSampling* in terms of all evaluation metrics: Recall, Precision, Specificity, F1, MCC, and AUC. These methods are categorized into multiple groups using the Scott-Knott ESD test, often resulting in more than six groups. Darker colors indicate higher-ranked groups, with each color representing two groups (i.e., the darkest purple denotes the first and second groups, moderate purple indicates the third and fourth groups, and the lightest purple represents the fifth and sixth groups). If some data resampling methods fall outside the top six groups but significantly outperform *NoSampling*, the results of these data resampling methods will be bolded.

In general, our analysis reveals consistent patterns in the performance of ten data resampling methods across different DLLAD approaches in terms of comprehensive metrics F1, MCC, and AUC. For instance, in Table 6.4, oversampling methods

Chapter 6. On the Influence of Data Resampling for Deep Learning-Based Log Anomaly Detection: Insights and Recommendations

Table 6.4 The Recall, Precision, Specificity, F1-score, MCC, and AUC values of **CNN** when employing various data resampling methods (i.e., NoSampling (NS), SMOTE (SMO), ADASYN (ADA), NearMiss (NM), InstanceHardnessThreshold (IHT), SMOTEENN (SE), SMOTETomek (ST), RandomOverSampling in the feature space (ROS_F), RandomUnderSampling in the feature space (RUS_F), RandomOverSampling applied to raw data (ROS_R), and RandomUnderSampling applied to raw data (RUS_R)) to the three datasets, each with the three different window sizes. Darker cells signify superior performance, while various colors denote statistical significance among data resampling methods for each evaluation metric (determined by the Scott-Knott ESD test with a p -value <0.05), as observed in the subsequent tables.

Dataset	ws	Metric	NS	SMO	ADA	NM	IHT	SE	ST	ROS_F	RUS_F	ROS_R	RUS_R
BGL	20	R	0.948	0.963	0.962	0.956	0.967	0.961	0.962	0.962	0.960	0.961	0.961
		P	0.985	0.978	0.982	0.989	0.945	0.995	0.989	0.984	0.986	0.984	0.986
		S	0.999	0.999	0.999	0.999	0.996	1.000	0.999	0.999	0.999	0.992	0.999
		F1	0.966	0.970	0.972	0.972	0.956	0.977	0.975	0.973	0.973	0.972	0.974
		MCC	0.964	0.968	0.970	0.970	0.953	0.976	0.973	0.971	0.971	0.972	0.972
		AUC	0.977	0.982	0.978	0.978	0.974	0.977	0.979	0.984	0.974	0.984	0.983
	50	R	0.907	0.910	0.912	0.898	0.944	0.896	0.915	0.931	0.915	0.918	0.899
		P	0.708	0.853	0.877	0.895	0.192	0.882	0.761	0.538	0.205	0.885	0.885
		S	0.965	0.987	0.990	0.991	0.684	0.990	0.971	0.916	0.717	0.990	0.991
		F1	0.787	0.880	0.894	0.896	0.319	0.888	0.820	0.665	0.335	0.901	0.892
		MCC	0.779	0.871	0.886	0.888	0.341	0.880	0.814	0.665	0.352	0.893	0.883
		AUC	0.919	0.941	0.959	0.952	0.824	0.937	0.943	0.929	0.857	0.944	0.951
	100	R	0.916	0.930	0.900	0.911	0.937	0.894	0.902	0.927	0.901	0.907	0.898
		P	0.362	0.857	0.832	0.923	0.218	0.885	0.665	0.905	0.798	0.928	0.613
		S	0.827	0.996	0.983	0.993	0.681	0.989	0.920	0.991	0.977	0.993	0.945
		F1	0.510	0.891	0.865	0.917	0.353	0.888	0.729	0.916	0.844	0.917	0.727
		MCC	0.509	0.889	0.852	0.909	0.359	0.878	0.722	0.908	0.831	0.910	0.713
		AUC	0.903	0.962	0.928	0.958	0.837	0.953	0.912	0.954	0.929	0.953	0.907
TB	20	R	0.495	0.495	0.532	0.532	0.550	0.000	0.495	0.568	0.541	0.550	0.495
		P	0.443	0.821	0.426	0.123	0.185	1.000	0.821	0.154	0.230	0.825	0.817
		S	0.999	1.000	0.999	0.997	0.998	1.000	1.000	0.998	0.998	1.000	1.000
		F1	0.441	0.618	0.443	0.198	0.274	0.000	0.618	0.241	0.315	0.659	0.613
		MCC	0.454	0.637	0.460	0.253	0.316	0.000	0.638	0.293	0.347	0.673	0.634
		AUC	0.866	0.840	0.873	0.880	0.894	0.545	0.839	0.894	0.865	0.881	0.843
	50	R	0.460	0.379	0.414	0.000	0.391	0.000	0.345	0.402	0.000	0.345	0.000
		P	0.169	0.800	0.315	0.000	0.255	0.000	0.537	0.481	0.000	0.923	0.000
		S	0.997	1.000	0.999	1.000	0.998	1.000	0.999	0.999	1.000	1.000	1.000
		F1	0.247	0.490	0.355	0.000	0.299	0.000	0.383	0.376	0.000	0.501	0.000
		MCC	0.277	0.536	0.359	0.000	0.309	0.000	0.410	0.406	0.000	0.563	0.000
		AUC	0.812	0.799	0.826	0.713	0.816	0.638	0.793	0.810	0.725	0.841	0.738
	100	R	0.304	0.290	0.348	0.043	0.261	0.000	0.319	0.348	0.000	0.348	0.000
		P	0.213	0.326	0.447	0.333	0.092	0.000	0.465	0.463	0.000	0.600	0.000
		S	0.997	0.998	0.999	1.000	0.996	1.000	0.998	0.999	1.000	0.999	1.000
		F1	0.241	0.276	0.391	0.077	0.136	0.000	0.334	0.397	0.000	0.420	0.000
		MCC	0.247	0.290	0.393	0.120	0.153	0.000	0.360	0.400	0.000	0.444	0.000
		AUC	0.786	0.788	0.776	0.736	0.786	0.656	0.788	0.801	0.733	0.802	0.730
Spirit	20	R	0.815	0.822	0.814	0.822	0.851	0.614	0.823	0.826	0.808	0.823	0.800
		P	0.797	0.922	0.864	0.797	0.482	0.845	0.934	0.975	0.975	0.991	0.946
		S	0.999	1.000	0.999	0.999	0.995	0.999	1.000	1.000	1.000	1.000	1.000
		F1	0.806	0.868	0.837	0.806	0.615	0.646	0.875	0.894	0.884	0.899	0.866
		MCC	0.805	0.869	0.837	0.807	0.638	0.681	0.876	0.897	0.887	0.902	0.869
		AUC	0.967	0.969	0.968	0.968	0.974	0.963	0.969	0.971	0.973	0.972	0.971
	50	R	0.893	0.907	0.888	0.908	0.963	0.065	0.893	0.935	0.903	0.908	0.889
		P	0.582	0.677	0.769	0.415	0.243	0.080	0.592	0.873	0.510	0.786	0.618
		S	0.991	0.994	0.996	0.983	0.959	0.990	0.992	0.998	0.988	0.997	0.992
		F1	0.704	0.774	0.823	0.570	0.388	0.071	0.712	0.903	0.652	0.842	0.728
		MCC	0.716	0.780	0.823	0.607	0.472	0.060	0.723	0.902	0.673	0.842	0.737
		AUC	0.996	0.997	0.997	0.993	0.978	0.921	0.996	0.998	0.995	0.998	0.997
	100	R	0.909	0.919	0.919	0.899	0.961	0.888	0.908	0.931	0.907	0.925	0.924
		P	0.726	0.853	0.771	0.510	0.311	0.623	0.657	0.904	0.784	0.880	0.803
		S	0.991	0.996	0.993	0.977	0.945	0.986	0.988	0.997	0.994	0.997	0.994
		F1	0.807	0.882	0.838	0.649	0.470	0.730	0.762	0.917	0.841	0.902	0.858
		MCC	0.807	0.881	0.837	0.666	0.530	0.735	0.765	0.915	0.839	0.899	0.857
		AUC	0.995	0.998	0.998	0.992	0.976	0.994	0.997	0.999	0.988	0.998	0.988

6.3 Results and Analysis

Table 6.5 The Recall, Precision, Specificity, F1-score, MCC, and AUC values of **LogRobust** when employing various data resampling methods to the three datasets, each with the three different window sizes.

Dataset	ws	Metric	NS	SMO	ADA	NM	IHT	SE	ST	ROS_F	RUS_F	ROS_R	RUS_R
BGL	20	R	0.949	0.958	0.955	0.951	0.952	0.952	0.956	0.938	0.942	0.951	0.948
		P	0.860	0.885	0.889	0.882	0.858	0.887	0.889	0.892	0.892	0.912	0.904
		S	0.989	0.992	0.992	0.991	0.989	0.992	0.992	0.992	0.992	0.994	0.993
		F1	0.903	0.920	0.921	0.915	0.903	0.918	0.921	0.914	0.916	0.931	0.925
		MCC	0.897	0.915	0.916	0.910	0.897	0.913	0.916	0.909	0.911	0.926	0.920
		AUC	0.970	0.985	0.988	0.980	0.974	0.983	0.982	0.974	0.975	0.980	0.982
	50	R	0.911	0.901	0.920	0.871	0.940	0.944	0.904	0.919	0.901	0.901	0.898
		P	0.709	0.895	0.879	0.648	0.565	0.821	0.823	0.891	0.872	0.902	0.896
		S	0.968	0.992	0.990	0.953	0.942	0.983	0.983	0.991	0.989	0.992	0.992
		F1	0.792	0.898	0.899	0.726	0.706	0.877	0.859	0.905	0.886	0.901	0.897
		MCC	0.783	0.890	0.891	0.718	0.703	0.869	0.849	0.897	0.877	0.893	0.889
		AUC	0.960	0.966	0.966	0.964	0.962	0.960	0.942	0.966	0.965	0.963	0.966
	100	R	0.903	0.928	0.878	0.917	0.945	0.887	0.908	0.889	0.900	0.936	0.887
		P	0.793	0.892	0.908	0.870	0.896	0.957	0.914	0.900	0.900	0.894	0.896
		S	0.978	0.989	0.992	0.987	0.978	0.990	0.996	0.992	0.990	0.989	0.990
		F1	0.844	0.909	0.892	0.893	0.868	0.892	0.932	0.901	0.900	0.915	0.892
		MCC	0.831	0.901	0.882	0.883	0.858	0.882	0.926	0.892	0.890	0.907	0.882
		AUC	0.954	0.963	0.961	0.959	0.961	0.961	0.971	0.967	0.964	0.966	0.961
TB	20	R	0.766	0.459	0.641	0.775	0.811	0.000	0.775	0.802	0.703	0.652	0.529
		P	0.084	0.708	0.218	0.108	0.127	0.000	0.111	0.141	0.313	0.285	0.352
		S	0.994	1.000	0.998	0.995	0.996	1.000	0.995	0.996	0.997	0.998	0.998
		F1	0.151	0.532	0.313	0.189	0.220	0.000	0.193	0.239	0.368	0.376	0.371
		MCC	0.252	0.557	0.364	0.286	0.319	0.000	0.290	0.335	0.425	0.415	0.403
		AUC	0.915	0.849	0.903	0.920	0.933	0.544	0.916	0.922	0.905	0.685	0.870
	50	R	0.609	0.529	0.598	0.347	0.644	0.000	0.540	0.552	0.425	0.598	0.425
		P	0.317	0.804	0.433	0.647	0.236	0.000	0.271	0.404	0.636	0.852	0.805
		S	0.998	1.000	0.999	1.000	0.996	1.000	0.998	0.998	1.000	1.000	1.000
		F1	0.415	0.637	0.481	0.450	0.333	0.000	0.355	0.444	0.497	0.703	0.554
		MCC	0.437	0.651	0.497	0.473	0.379	0.000	0.378	0.459	0.513	0.713	0.583
		AUC	0.861	0.881	0.864	0.847	0.885	0.524	0.872	0.854	0.860	0.870	0.865
	100	R	0.449	0.464	0.435	0.536	0.565	0.029	0.485	0.435	0.507	0.464	0.420
		P	0.277	0.436	0.510	0.216	0.146	0.667	0.374	0.597	0.469	0.655	0.488
		S	0.997	0.998	0.999	0.995	0.992	1.000	0.998	0.999	0.999	0.999	0.999
		F1	0.340	0.429	0.457	0.297	0.232	0.056	0.418	0.501	0.484	0.526	0.450
		MCC	0.350	0.438	0.463	0.331	0.284	0.139	0.422	0.507	0.485	0.541	0.451
		AUC	0.826	0.868	0.846	0.851	0.848	0.727	0.859	0.865	0.864	0.831	0.837
Spirit	20	R	0.892	0.910	0.914	0.910	0.937	0.791	0.913	0.913	0.915	0.918	0.915
		P	0.973	0.991	0.990	0.836	0.510	0.547	0.975	0.993	0.996	0.990	0.989
		S	1.000	1.000	1.000	0.999	0.995	0.992	1.000	1.000	1.000	1.000	1.000
		F1	0.930	0.949	0.950	0.862	0.660	0.542	0.943	0.951	0.954	0.953	0.950
		MCC	0.931	0.950	0.951	0.867	0.689	0.599	0.943	0.951	0.954	0.953	0.951
		AUC	0.991	0.990	0.993	0.988	0.991	0.980	0.992	0.993	0.994	0.993	0.992
	50	R	0.910	0.903	0.914	0.896	0.974	0.899	0.903	0.907	0.910	0.925	0.907
		P	0.805	0.949	0.763	0.736	0.218	0.717	0.893	0.957	0.884	0.905	0.917
		S	0.997	0.999	0.996	0.996	0.952	0.995	0.999	0.999	0.998	0.999	0.999
		F1	0.832	0.925	0.832	0.808	0.356	0.798	0.898	0.931	0.897	0.915	0.912
		MCC	0.854	0.925	0.833	0.809	0.449	0.800	0.897	0.930	0.896	0.914	0.911
		AUC	0.990	0.990	0.990	0.989	0.967	0.989	0.990	0.990	0.990	0.991	0.990
	100	R	0.918	0.923	0.920	0.925	0.981	0.765	0.913	0.939	0.923	0.909	0.916
		P	0.863	0.945	0.904	0.609	0.330	0.725	0.644	0.972	0.895	0.983	0.950
		S	0.994	0.999	0.996	0.984	0.949	0.991	0.987	0.999	0.997	1.000	0.999
		F1	0.889	0.933	0.911	0.733	0.493	0.717	0.753	0.955	0.908	0.945	0.932
		MCC	0.885	0.932	0.908	0.742	0.553	0.725	0.759	0.954	0.906	0.944	0.931
		AUC	0.996	0.999	0.997	0.993	0.982	0.994	0.995	0.998	0.997	0.997	0.996

(*SMOTE* and *ADASYN*) and the oversampling method ROS_R applied to raw data, significantly enhance the overall performance of CNN across datasets, with improvements ranging from 0.1% to 102.7% in F1, 0.5% to 103.2% in MCC, and

Chapter 6. On the Influence of Data Resampling for Deep Learning-Based Log Anomaly Detection: Insights and Recommendations

Table 6.6 The Recall, Precision, Specificity, F1-score, MCC, and AUC values of **NeuralLog** when employing various data resampling methods to the three datasets, each with the three different window sizes.

Dataset	ws	Metric	NS	SMO	ADA	NM	IHT	SE	ST	ROS_F	RUS_F	ROS_R	RUS_R
BGL	20	R	0.896	0.951	0.929	0.842	0.766	0.867	0.875	0.939	0.897	0.937	0.925
		P	0.852	0.932	0.908	0.166	0.818	0.908	0.925	0.894	0.958	0.923	0.910
		S	0.989	0.995	0.994	0.691	0.987	0.994	0.995	0.992	0.997	0.995	0.994
		F1	0.872	0.941	0.918	0.275	0.788	0.886	0.899	0.915	0.926	0.930	0.917
		MCC	0.864	0.937	0.913	0.282	0.776	0.880	0.893	0.910	0.922	0.925	0.912
		AUC	0.943	0.971	0.956	0.767	0.877	0.930	0.935	0.966	0.947	0.966	0.955
	50	R	0.627	0.753	0.891	0.755	0.732	0.645	0.675	0.899	0.680	0.880	0.807
		P	0.872	0.825	0.840	0.284	0.740	0.900	0.888	0.844	0.956	0.897	0.903
		S	0.991	0.985	0.984	0.621	0.979	0.992	0.990	0.986	0.998	0.992	0.992
		F1	0.721	0.770	0.854	0.325	0.735	0.739	0.747	0.869	0.794	0.888	0.837
		MCC	0.718	0.763	0.849	0.297	0.715	0.740	0.749	0.860	0.794	0.879	0.836
		AUC	0.809	0.869	0.937	0.688	0.856	0.819	0.833	0.943	0.839	0.936	0.899
	100	R	0.598	0.892	0.828	0.640	0.874	0.755	0.713	0.842	0.520	0.878	0.656
		P	0.671	0.715	0.831	0.323	0.190	0.799	0.887	0.883	0.808	0.879	0.904
		S	0.878	0.963	0.982	0.601	0.650	0.980	0.991	0.989	0.985	0.989	0.993
		F1	0.496	0.787	0.818	0.336	0.312	0.767	0.789	0.859	0.603	0.878	0.759
		MCC	0.511	0.774	0.807	0.259	0.300	0.751	0.778	0.848	0.608	0.866	0.752
		AUC	0.738	0.919	0.916	0.620	0.763	0.890	0.852	0.905	0.753	0.927	0.829
TB	20	R	0.772	0.862	0.561	0.821	0.683	0.496	0.691	0.772	0.683	0.756	0.740
		P	0.758	0.800	0.852	0.717	0.825	0.886	0.869	0.842	0.886	0.942	0.883
		S	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
		F1	0.760	0.827	0.676	0.765	0.738	0.635	0.770	0.805	0.771	0.838	0.805
		MCC	0.762	0.829	0.691	0.766	0.745	0.662	0.775	0.806	0.778	0.843	0.808
		AUC	0.856	0.931	0.780	0.910	0.841	0.748	0.845	0.886	0.841	0.886	0.878
	50	R	0.730	0.854	0.805	0.854	0.748	0.537	0.748	0.813	0.813	0.879	0.715
		P	0.469	0.788	0.797	0.700	0.803	0.880	0.671	0.808	0.603	0.986	0.935
		S	0.999	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.999	1.000	1.000
		F1	0.571	0.812	0.797	0.769	0.770	0.667	0.686	0.808	0.683	0.928	0.811
		MCC	0.585	0.816	0.799	0.773	0.773	0.687	0.697	0.809	0.695	0.930	0.818
		AUC	0.865	0.927	0.902	0.927	0.874	0.768	0.874	0.906	0.906	0.939	0.899
	100	R	0.756	0.756	0.911	0.683	0.732	0.512	0.618	0.805	0.780	0.813	0.732
		P	0.470	0.697	0.497	0.583	0.698	0.778	0.790	0.750	0.681	0.828	0.882
		S	0.999	1.000	0.999	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
		F1	0.579	0.718	0.641	0.629	0.714	0.618	0.642	0.776	0.727	0.820	0.800
		MCC	0.596	0.722	0.671	0.631	0.714	0.631	0.672	0.777	0.729	0.820	0.803
		AUC	0.828	0.878	0.955	0.841	0.866	0.756	0.832	0.902	0.890	0.906	0.866
Spirit	20	R	0.899	0.919	0.929	0.950	0.940	0.881	0.925	0.939	0.920	0.919	0.924
		P	0.899	0.969	0.902	0.380	0.510	0.976	0.945	0.928	0.800	0.976	0.889
		S	0.999	1.000	0.999	0.990	0.995	1.000	1.000	1.000	0.999	1.000	0.999
		F1	0.895	0.943	0.915	0.534	0.661	0.925	0.935	0.933	0.855	0.946	0.905
		MCC	0.897	0.943	0.915	0.591	0.690	0.927	0.935	0.933	0.857	0.947	0.905
		AUC	0.949	0.959	0.964	0.970	0.968	0.940	0.962	0.969	0.960	0.960	0.962
	50	R	0.931	0.913	0.937	0.944	0.946	0.889	0.919	0.932	0.933	0.923	0.919
		P	0.864	0.975	0.921	0.446	0.588	0.969	0.957	0.947	0.921	0.997	0.997
		S	0.999	1.000	1.000	0.993	0.996	1.000	1.000	1.000	1.000	1.000	1.000
		F1	0.896	0.943	0.929	0.603	0.724	0.926	0.937	0.939	0.927	0.959	0.957
		MCC	0.896	0.943	0.928	0.644	0.743	0.927	0.937	0.939	0.926	0.959	0.957
		AUC	0.965	0.962	0.968	0.968	0.971	0.944	0.960	0.966	0.966	0.962	0.961
	100	R	0.938	0.930	0.937	0.829	0.951	0.853	0.937	0.923	0.926	0.937	0.937
		P	0.800	0.985	0.989	0.810	0.556	0.938	0.927	0.989	0.974	0.996	0.947
		S	0.999	1.000	1.000	1.000	0.996	1.000	1.000	1.000	1.000	1.000	1.000
		F1	0.862	0.957	0.962	0.819	0.702	0.893	0.932	0.955	0.950	0.966	0.942
		MCC	0.865	0.957	0.962	0.819	0.726	0.894	0.932	0.955	0.950	0.966	0.941
		AUC	0.964	0.965	0.968	0.915	0.973	0.926	0.968	0.961	0.963	0.968	0.964

0.1% to 6.6% in AUC. Table 6.5 illustrates that utilizing *SMOTE*, ROS_F , and RUS_F within the feature space, in addition to the resampling methods applied directly to raw data (ROS_R and RUS_R), leads to significant performance improvements for LogRobust

across all datasets. These methods demonstrate notable enhancements in F1 (improved by 1.3% to 252.3%), MCC (improved by 1.3% to 121.0%), and AUC (improved by 0.3% to 5.1%). Additionally, in many cases, *ADASYN* and *SMOTETomek* significantly perform better than *NoSampling*. Similarly, Table 6.6 highlights the effectiveness of data resampling for NeuralLog. Most data resampling methods, including *SMOTE*, *SMOTETomek*, ROS_F , ROS_R , and RUS_R , can achieve significantly better performance than *NoSampling* across all datasets with improvements ranging from 1.1% to 77.0% in F1, 1.8% to 69.5% in MCC, and 0.1% to 25.6% in AUC. In summary, *SMOTE*, ROS_F , ROS_R , and RUS_R applied to DLLAD approaches significantly outperform *NoSampling*. Additionally, improvements are observed in other metrics such as Recall and Precision.

In addition, we observe that employing ROS_R on all three DLLAD approaches consistently yields superior performance. Specifically, ROS_R ranks within the top two groups (represented by the darkest purple color) in all 27 cases (3 DLLAD approaches $\times$ 3 datasets $\times$ 3 window sizes) in terms of at least two comprehensive metrics. *SMOTE* emerges as the second recommended data resampling method, demonstrating significantly better performance than *NoSampling* in 26 out of 27 cases across all datasets in terms of at least two comprehensive metrics. Similarly, ROS_F and RUS_R outperform *NoSampling* in 25 out of 27 cases, followed by *ADASYN* in 24 out of 27 cases. In contrast, *NearMiss*, *IHT*, and *SMOTEENN* exhibit poor performance, succeeding in only 10 out of 27, 4 out of 27, and 13 out of 27 cases, respectively.

Finding 5: Employing *SMOTE*, ROS_F , ROS_R , and RUS_R to alleviate the class imbalance can yield better results compared to *NoSampling* in DLLAD. ↗

Finding 6: Data resampling on the raw data (ROS_R and RUS_R) tends to outperform data resampling in the feature space (including the advanced hybrid one *SMOTETomek*), particularly in the case of undersampling. ↗

Finding 7: In general, *oversampling* exhibits better performance compared to *undersampling* and *hybrid sampling*.

In datasets characterized by moderate imbalance, such as BGL and Spirit, several data resampling methods contribute to improved performance in DLLAD approaches. Given the consistent results between F1 and MCC, and the minimal changes in AUC in certain cases, our analysis primarily focuses on the outcomes of MCC, a recommended symmetric metric. For the BGL dataset with a *ws* of 20, CNN achieves high performance with an MCC of 0.964 (as shown in Table 6.4). Most data resampling

Chapter 6. On the Influence of Data Resampling for Deep Learning-Based Log Anomaly Detection: Insights and Recommendations

methods, excluding *IHT*, contribute to a slight improvement of 0.4%–1.2%. When the ws is set to 100, the impact of data resampling methods becomes more significant, with enhancements ranging from 40.1% to 78.8%. The Recall is increased by the reduction in false positives, leading to a higher percentage of accurately predicted anomalies and an overall enhancement in the comprehensive metric MCC. With $ws=50$, *IHT*, ROS_F , and RUS_F perform inferiorly to *NoSampling*, while others improve *NoSampling* by 5.5%–14.6%. Similarly, for the same dataset with $ws=20$, most data resampling methods, except for *IHT*, enhance the performance of LogRobust (Table 6.5), with improvements ranging from 1.3% to 3.2%. At $ws=50$, aside from undersampling methods applied to the feature space (i.e., *NearMiss* and *IHT*), all other methods positively impact the performance, with improvements of 8.4%–14.6%. At $ws=100$, all data resampling methods enhance *NoSampling* with an increase of 3.2%–11.4%. In the case of NeuralLog, data resampling methods consistently show improvement trends, as shown in Table 6.6. Except for *NearMiss* and *IHT*, all other resampling methods lead to improvements ranging from 1.9% to 69.5% across three window sizes.

For the Spirit dataset with $ws=20$, Table 6.4 illustrates that most data resampling methods, except for *NearMiss*, *IHT*, and *SMOTEENN*, contribute to an improvement in the performance of CNN, with enhancements ranging from 4.0% to 12.0% in MCC. With $ws=50$, except for *NearMiss*, *IHT*, *SMOTEENN*, and RUS_R , other methods show positive effects, resulting in improvements of 1.0% to 26.0%. At $ws=100$, except for *NearMiss*, *IHT*, *SMOTEENN*, and *SMOTETomek*, other methods enhance *NoSampling* by 3.7% to 13.4%. Table 6.5 indicates that data resampling on LogRobust has a comparable impact to data resampling on CNN. With the exception of undersampling and hybrid sampling methods applied within the feature space, other methods excluding *ADASYN* contribute to an increase in *NoSampling* by 1.3% to 8.9% across three window sizes. When applying data resampling to NeuralLog, only *NearMiss*, *IHT*, and RUS_F do not enhance its performance across window sizes, as demonstrated in Table 6.6. Other data resampling methods show improvements of 0.9% to 11.7% compared to *NoSampling*.

In the highly imbalanced Thunderbird dataset, both ROS_R and *SMOTE* demonstrate overall performance enhancements for DLLAD approaches, with ROS_R showing statistically significant improvements across all cases. When applying data resampling methods to CNN on this dataset, unexpected results appear. Undersampling methods, such as RUS_F , RUS_R , *NearMiss*, as well as the hybrid sampling method *SMOTEENN*, uniformly cause the CNN to classify all log sequences as normal sequences, resulting in performance metrics (i.e., Recall, Precision, F1, and MCC) being reduced to 0.

Conversely, ROS_R improves *NoSampling* by 48.2%, 103.2%, and 79.8% for ws set to 20, 50, and 100, respectively. Surprisingly, a statistically significant difference in CNN performance is only observed with *SMOTETomek* and ROS_R . Table 6.5 demonstrates that when applying data resampling to LogRobust on this dataset, all data resampling methods except for *NearMiss*, *IHT*, *SMOTEENN*, and *SMOTETomek* yield improvements ranging from 5.0% to 121.0% across three window sizes. In contrast to the effects observed on CNN, it is worth noting that undersampling, oversampling, and hybrid sampling methods except for *ADASYN*, *IHT*, and *SMOTEENN*, can positively impact NeuralLog (as shown in Table 6.6). Particularly, random sampling applied to raw data (i.e., ROS_R and RUS_R) substantially enhances the MCC of *NoSampling* by 6.0% to 59.0%. In addition, concerning other metrics such as Recall and Precision, data resampling methods (except for *NearMiss*, *IHT*, and *SMOTEENN*) significantly outperform *NoSampling* across all datasets in most cases.

Finding 8: In cases of severe class imbalance (e.g., the abnormal proportion is less than 1%), ROS_R consistently enhances the effectiveness of DLLAD approaches. In cases of moderate class imbalance, *SMOTE*, ROS_F , ROS_R , and RUS_R exhibit effectiveness improvements across DLLAD approaches. ↗

Furthermore, there are other noteworthy observations from the analysis. Some undersampling methods (i.e., *NearMiss* and *IHT*) might not perform well in many cases, particularly in the dataset with severe class imbalance. The limited performance of these methods can be attributed to the significant reduction of the majority class while addressing class imbalance. This reduction poses a risk of losing vital information present in the log data. Hybrid sampling methods like *SMOTEENN* attempt to address class imbalance by generating synthetic abnormal sequences and subsequently eliminating normal log sequences that share overlapping features with another class. Similarly, *SMOTEENN* cannot effectively improve the performance of DLLAD approaches. One potential reason for this limitation is that removing normal log sequences based on overlapping features between the synthetic abnormal sequences (which may contain noise) and the normal data may also result in the loss of crucial information. In contrast, oversampling methods, which focus on generating additional log sequences of the minority class, effectively mitigate the class imbalance issue. While this process may introduce some level of noise due to the replication and generation of abnormal log sequences, it seems to be a more effective strategy for preserving crucial information within log data. These observations emphasize the

Chapter 6. On the Influence of Data Resampling for Deep Learning-Based Log Anomaly Detection: Insights and Recommendations

trade-off between addressing class imbalance and retaining critical data information when applying the data resampling methods in DLLAD.

Finding 9: Undersampling methods *NearMiss* and *IHT* are often ineffective in addressing the data imbalance problem. In addition, the hybrid method *SMOTEENN* generally does not improve the performance of DLLAD approaches. ↘

Table 6.7 An example of an **abnormal** log sequence from the Spirit dataset with $ws=20$, where the second log event is highlighted in orange as it represents an anomaly (left), and the top five tokens in the log sequence contributing to the DLLAD model’s correct prediction are highlighted in orange (right).

#	Log event	Top five tokens
1	publickey for < * > from < * > port < * > ssh< * >	NoSampling ✗: publickey, opened, closed, port, user
2	LANai is not running. Allowing port=< * > open for debugging	SMOTE ✔: publickey, debugging, LANai, opened, user
3	NOTICE: < * >	
4	closed for user < * >	
5	opened for user < * > by < * >	ADASYN ✔: publickey, debugging, opened, user, ssh
6	publickey for < * > from < * > port < * > ssh< * >	
7	opened for user < * > by < * >	NearMiss ✔: publickey, opened, closed, user, debugging
8	publickey for < * > from < * > port < * > ssh< * >	
9	opened for user < * > by < * >	IHT ✔: publickey, debugging, LANai, opened, user
10	publickey for < * > from < * > port < * > ssh< * >	
11	opened for user < * > by < * >	SMOTEENN ✗: publickey, closed, opened, user, NOTICE
12	publickey for < * > from < * > port < * > ssh< * >	
13	opened for user < * > by < * >	SMOTETomek ✔: publickey, opened, NOTICE, debugging, closed
14	publickey for < * > from < * > port < * > ssh< * >	
15	opened for user < * > by < * >	ROS_F ✔: publickey, debugging, NOTICE, LANai, closed
16	publickey for < * > from < * > port < * > ssh< * >	
17	opened for user < * > by < * >	RUS_F ✔: publickey, debugging, LANai, NOTICE, closed
18	publickey for < * > from < * > port < * > ssh< * >	
19	opened for user < * > by < * >	ROS_R ✔: debugging, port, open, publickey, NOTICE
20	publickey for < * > from < * > port < * > ssh< * >	RUS_R ✔: debugging, NOTICE, port, user, publickey

6.4 Discussion

6.4.1 Why Do(not) Data Resampling Methods Work

In RQ3, we investigate the impact of various data resampling methods on the performance of DLLAD approaches. To enhance the interpretability of their performance, we utilize Local Interpretable Model-agnostic Explanations (LIME) [243], which is a widely-used model-agnostic explainable algorithm for explaining deep learning models in software engineering [244–249]. LIME can identify the most important tokens in a log sequence that contribute to the DLLAD model’s prediction.

Table 6.8 An example of a **normal** log sequence from the Thunderbird dataset with $ws=20$, along with the top five tokens in the log sequence contributing to the DLLAD model’s prediction (right).

#	Log event	Top five tokens
1	client does not accept options	NoSampling ✗: client, no, free, leases, options
2	client does not accept options	SMOTE ✓: client, options, Temperature, changed, stratum
3	client does not accept options	ADASYN ✓: client, options, Temperature, Celsius, stratum
4	client does not accept options	NearMiss ✗: client, options, no, free, leases
5	client does not accept options	IHT ✓: client, options, Temperature, stratum, network
6	from < * > via eth1: network < * > no free leases	SMOTEENN ✓: client, options, Celsius, Temperature, network
7	from < * > via eth1: network < * > no free leases	SMOTETomek ✓: client, options, Celsius, Temperature, stratum
8	/dev/sda, Temperature changed < * > Celsius to < * > Celsius since last report	ROS_F ✓: client, options, Temperature, Celsius, stratum
9	/dev/sda, Temperature changed < * > Celsius to < * > Celsius since last report	RUS_F ✓: client, options, Celsius, Temperature, stratum
10	to < * > stratum < * >	ROS_R ✓: client, options, Temperature, Celsius, stratum
11	to < * > stratum < * >	RUS_R ✓: client, options, Temperature, Celsius, stratum
12	to < * > stratum < * >	
13	/dev/sda, Temperature changed < * > Celsius to < * > Celsius since last report	
14	client does not accept options	
15	client does not accept options	
16	client does not accept options	
17	client does not accept options	
18	client does not accept options	
19	client does not accept options	
20	client does not accept options	

In Tables 6.7 and 6.8, we present two examples showcasing the LIME results for an actual abnormal and normal log sequence, both with and without employing data resampling methods. The first column denotes the number of log events in the log sequence, while the second column describes the specific content of each log event. The last column displays the top five tokens contributing to the LogRobust prediction with various data resampling methods. (1) The example in Table 6.7 is derived from the Spirit dataset with $ws=20$, where certain data resampling methods perform better than *NoSampling* in terms of Recall. The second log event, highlighted in orange, signifies its abnormal nature. This event signifies the attempt to open a port for debugging purposes, which is typically considered an abnormal operation in a production environment. Such an action may suggest a potential security risk or an effort to troubleshoot an issue. LogRobust with *NoSampling* erroneously predicts the actual abnormal log sequence as normal. Examining the top five tokens reveals none of them appearing in the abnormal log event, thereby failing to accurately capture any abnormal information. However, when employing data resampling methods except for *SMOTEENN*, LogRobust correctly predicts the abnormal sequence based on highlighted abnormal tokens. Notably, with *ROS_R*, three tokens overlap with the abnormal log event, suggesting that properly duplicated log sequences in the raw data could positively influence DLLAD models in learning real patterns for detecting anomalies. (2) The example in Table 6.8 is derived from the Thunderbird dataset with $ws=20$, where certain data resampling methods substantially improve the Precision values compared to *NoSampling*. LogRobust with *NoSampling* erroneously predicts the actual normal log event as abnormal. The top five tokens extracted by LogRobust

with *NoSampling* primarily focus on the sixth and seventh log events. These events describe the unavailability of an IP address for allocation. While this repetition of log messages may imply network configuration or capacity planning issues, it is important to consider that in a dynamically changing network environment, especially with a large number of temporary devices or clients, such occurrences may be anticipated. After employing data resampling methods, except for *NearMiss*, the LogRobust model shifts its attention to different tokens, such as “Temperature”, “changed”, and “Celsius”, and predicts the actual normal log event as normal. In summary, certain data resampling methods can prioritize crucial tokens in log sequences, aiding DLLAD models in making more accurate predictions than *NoSampling*. Consequently, DLLAD models utilizing data resampling methods demonstrate improved prediction capabilities for both abnormal and normal log sequences. These advancements may result from the enhanced distribution of training data achieved through data resampling methods, enabling the model to better distinguish between normal and abnormal log sequences based on learned patterns.

6.4.2 Implications of Our Findings

The analysis of the experiments demonstrated the superiority of *SMOTE*, ROS_F , ROS_R , and RUS_R over other data resampling methods in most cases. The choice of window size and DLLAD model significantly influence the comparative outcomes. This research carries several practical implications for practitioners.

(1) Efficiency prioritization: Practitioners typically aim to efficiently train DLLAD models to identify log anomalies within a short timeframe. In this regard, CNN and LogRobust require less training time, and larger window sizes can notably reduce costly processing time. Additionally, applying *ROS* and *RUS* methods on raw data for data duplication and removal before embedding takes considerably less time than data resampling methods in feature spaces. Based on our experimental findings, LogRobust outperforms CNN overall. Hence, employing ROS_R and RUS_R with LogRobust is recommended for efficient prediction.

(2) Effectiveness prioritization: For non-urgently training DLLAD models to detect log anomalies, practitioners prioritize high accuracy in predicted results. In datasets with smaller window sizes and more balanced distributions (e.g., BGL and Spirit), the difference between LogRobust and NeuralLog is not significant. However, for highly imbalanced datasets (e.g., Thunderbird), NeuralLog noticeably outperforms

LogRobust. Thus, it is advisable to adopt these data resampling methods (*SMOTE*, *ROS_F*, *ROS_R*, and *RUS_R*) with NeuralLog for effective prediction.

6.5 Threats to Validity

6.5.1 Limited Models and Datasets

One potential concern pertains to our selection of DLLAD approaches, we adapted existing supervised approaches for our empirical investigation. This choice is motivated by several factors: Semi-supervised approaches leverage only a fraction of normal logs, whereas unsupervised methods assume datasets lack labels, diverging from our fully supervised data scenario. Furthermore, some unsupervised and semi-supervised approaches share a model structure similar to NeuralLog, as observed in LAnoBERT [115] and Hades [14]. Previous empirical studies [11, 12] have consistently shown inferior performance of unsupervised and semi-supervised approaches compared to supervised ones. Hence, we deliberately include the latest supervised approaches as our evaluated DLLAD approaches. Another concern arises from the limited availability of datasets. Given that there are currently only several public datasets available (i.e., HDFS, BGL, Thunderbird, and Spirit), approaches have already demonstrated good performance on HDFS, we have chosen to exclusively use the remaining three datasets for our experimental analysis. In future work, we intend to investigate the generalizability of our conclusions across more datasets.

6.5.2 Hyperparameter Settings

The selection of an appropriate resampling ratio of normal to abnormal data constitutes a threat when assessing the effectiveness of data resampling methods. Considering numerous data resampling methods and the impracticality of evaluating all of them across diverse datasets with exhaustive resampling ratio variations, it is crucial to systematically choose the resampling ratio. To address this challenge, we introduce a standardized quarter-based unit for a consistent benchmark across various data resampling methods. On this basis, the general conclusions obtained aid researchers in narrowing their focus for subsequent study phases.

6.5.3 Generalizability

To enhance the generalizability of our findings, we intentionally select three distinct categories encompassing a total of ten data resampling methods. These data resampling methods are systematically applied to three existing DLLAD approaches on publicly available datasets. Our objective does not compare the effectiveness of those DLLAD approaches but rather focuses on evaluating the capabilities of different data resampling methods applied to those approaches. As such, our findings aim to highlight general trends in how data resampling affects different DLLAD approaches, with the ultimate goal of offering valuable insights to inform future research endeavors.

6.6 Chapter Summary

In this chapter, our study represents a pioneering effort in comprehensively assessing the impact of ten data resampling methods on alleviating class imbalance in DLLAD. Through empirical analysis, we have derived several critical insights. Firstly, severe data imbalances, like in the Thunderbird dataset, often lead to poor performance in DLLAD approaches. Secondly, oversampling methods generally outperform both undersampling and hybrid sampling methods. Moreover, data resampling on raw data yields superior results compared to data resampling in the feature space. Notably, ROS_R exhibits outstanding performance, particularly in scenarios characterized by severe class imbalances. Thirdly, certain undersampling and hybrid sampling methods, such as *SMOTEENN* and *IHT*, show limited effectiveness in most cases. Furthermore, our exploration of different resampling ratios of normal to abnormal data provides actionable recommendations for optimizing the impact of data resampling on DLLAD approaches. We suggest generating more data for minority classes (oversampling) to significantly reduce class imbalance while removing less data from majority classes (undersampling). In summary, our findings provide a valuable roadmap for researchers seeking to address the issue of data imbalance in DLLAD. By adopting the recommended strategies for data resampling, researchers can enhance the performance and effectiveness of DLLAD approaches.

Data Availability. Our source-code and data are publicly available².

²<https://github.com/ResamplingDLLAD/ResamplingEmpirical>

Chapter 7

Conclusion and Future Work

This thesis introduces innovative text representation learning approaches aimed at overcoming challenges in four intelligent software engineering tasks: security bug report detection, bug report title generation, log anomaly detection, and class imbalance in log anomaly detection. The chapter wraps up with a summary of the contributions made and offers insightful recommendations for future research directions.

7.1 Thesis Contributions

In the area of security bug report detection, we introduce CASMS. It employs $tf - idf$ weighted CBOW for text representation and then applies the k -means clustering algorithm to group data. Within each cluster, a specific number of non-security bug reports closest to the centroid are chosen as representative samples. The Elbow method is utilized to determine the optimal number of clusters. Subsequently, the selected non-security samples and all security samples are fed into a CNN-BLSTM model to enhance representation learning. Through extensive experiments conducted on 44,940 bug reports, our proposed approach demonstrates its capability to effectively detect security bug reports and compete with baseline methods. It shows an improved recall of 10.33% to 36.24% and an enhanced g -measure of 4.09% to 24.26%, coupled with strong robustness.

In the area of bug report title generation, we propose AttSum. It employs an encoder-decoder framework featuring a pre-trained RoBERTa as the encoder and a stacked Transformer as the decoder. In addition, we incorporate a copy mechanism to learn some necessary tokens from the descriptive bug report body to the generated title. Through experiments conducted on 333,563 bug reports and subsequent analysis

Chapter 7. Conclusion and Future Work

using automatic and human evaluation methods, our approach demonstrates superiority over extractive and abstractive baselines. Specifically, we observe improvements of 3.4%–58.8% in ROUGE F1 and 7.7%–42.3% in BLEU scores. Furthermore, manual evaluation is conducted on 200 bug reports with titles generated by baselines and 50 low-quality bug reports with two titles (one generated by our approach and the other the original). The results reveal that AttSum outperforms baselines across all modalities, showcasing improvements of 1.9%–57.5% and statistically significant enhancements in low-quality titles. These findings demonstrate that our approach excels in generating more coherent and precise bug report titles.

In the area of log anomaly detection, we put forward AdaLog, which is a semi-supervised approach. It leverages the pre-trained BERT for log representation and uses k -means to roughly classify unlabeled and labeled training log data into distinct groups. Subsequently, utilizing the distances between unlabeled samples and cluster centroids, as well as the average distances between normal/abnormal samples and their respective centroids, our self-adaptive clustering method divides the calculation of the probability of each unlabeled sample being normal into twelve scenarios. Following this, samples with probabilities falling within the range $[0,1]$ are fed into the transformer model for representation learning and prediction. We conduct a comprehensive evaluation comparing AdaLog with SOTA approaches. The experimental results demonstrate its superiority, with AdaLog enhancing semi-supervised baselines by an average of 17.8% to 2489.8% in terms of F1 score. Additionally, AdaLog outperformed two supervised baselines in most cases, with an average improvement of 10.9% to 23.8% in F1 score. Furthermore, we recommend a reasonable undersampling ratio to further enhance performance.

Given the persistent challenge of class imbalance in log anomaly detection, we investigate the impact of alleviating this imbalance through data resampling methods on performance improvement. Initially, we assess the performance of representative approaches on three public datasets, revealing that severe data imbalance often leads to subpar results. Subsequently, we explore the effectiveness of various oversampling, undersampling, and hybrid sampling methods on these approaches. Our observations indicate that oversampling methods generally outperform both undersampling and hybrid sampling methods, and data resampling on the raw data yields superior results compared to resampling in the feature space. Moreover, we advise against certain undersampling and hybrid sampling methods, such as SMOTEENN and InstanceHardnessThreshold. Additionally, we recommend generating more minority data for oversampling and removing less majority data for undersampling to optimize

performance. These insights offer valuable guidance and strategic recommendations for leveraging data resampling to enhance the effectiveness of log anomaly detection.

7.2 Contributions to Software Engineering

These contributions collectively enhance software engineering practices by providing practical, validated tools and insights that address common challenges.

CASMS Approach for SBR Identification: The proposed CASMS approach aims to effectively and efficiently identify SBRs within bug tracking systems, enhancing the overall management and organization of bug reports. By accurately clustering similar reports, CASMS helps reduce redundancy, streamline the triage process, and improve prioritization, thus optimizing the workflow for developers and support teams.

AttSum for Generating High-Quality Bug Report Titles: AttSum, our proposed title generation method, has been validated through both automated metrics and manual evaluations, demonstrating its ability to produce high-quality bug report titles. This method can be directly integrated into bug tracking systems as a tool to assist bug reporters in drafting clear and informative titles. AttSum can act as a template or suggestion feature, guiding users to create consistent and descriptive titles, which can significantly enhance the readability and usability of bug reports.

AdaLog for Log Anomaly Detection: AdaLog is a semi-supervised approach designed to identify anomalies in log data, requiring only a subset of labeled logs. This makes it particularly suitable as a plug-in function for log analysis in monitoring tools, where labeled data is often scarce. By integrating AdaLog, log monitoring systems can improve their ability to detect irregularities and potential issues in real-time, thus supporting proactive maintenance and faster issue resolution.

Addressing Class Imbalance and Expanding Applications: The fourth contribution tackles the issue of class imbalance in log anomaly detection, offering strategies and insights that extend beyond the initial study's context. These approaches are valuable across various software engineering domains, including different types of anomaly detection, fraud detection, and other areas where imbalanced data presents significant challenges. This work lays the foundation for future research into advanced data balancing techniques, aiming to enhance the robustness and reliability of machine learning/deep learning models across diverse applications.

7.3 Limitations and Future Research Work

While our approaches for representation learning have yielded notable successes, there remain avenues for further enhancement based on our findings.

In the area of security bug report detection, our current methodology involves utilizing a *word2vec* model for text representation, employing a clustering method, and implementing a combined sequential model for semantic capture and representation learning from bug reports. Moving forward, we aim to streamline these processes by implementing a more generalized framework. This may entail integrating a pre-trained language model with a classification model to establish an end-to-end mapping between text representation and the learning classification objective. Such an approach would facilitate automation and potentially improve overall performance in bug report detection.

In the area of bug report title generation, our proposed deep attention-based summarization model, AttSum, utilizes a RoBERTa encoder for text representation and a stacked transformer decoder for text generation. Additionally, a copy mechanism is incorporated to address rare token occurrences. Large language models (LLMs), such as RoBERTa, play a crucial role in natural language processing due to their exceptional performance. We employ a common strategy of fine-tuning these models using downstream datasets, yielding promising results. In the future, we aim to explore methods for reducing tuning costs while preserving the effectiveness of LLMs, especially considering their increasing size and the volume of training data. To this end, we consider in-context learning as a novel approach, leveraging LLMs with demonstrations to perform downstream tasks without requiring parameter updates.

In the area of log anomaly detection, we present a semi-supervised approach AdaLog, which incorporates a self-adaptive clustering method for representation features. Our focus in future work will be on improving the robustness of our approach to resist external attacks. Previous studies [250–252] have pointed out that data-driven deep learning models are vulnerable to perturbations in input data, making them susceptible to adversarial attacks. While AdaLog leverages a transformer, a deep learning model, for anomaly prediction, its semi-supervised nature and utilization of a self-adaptive clustering technique mitigate sensitivity to input variations. However, we recognize the importance of integrating adversarial attack techniques to bolster the overall effectiveness and resilience of AdaLog. This enhancement will facilitate its application across various domains within the log anomaly detection field.

7.3 Limitations and Future Research Work

In the area of log anomaly detection, our empirical investigation has shed light on the significance of data resampling in addressing class imbalance and its subsequent impact on performance. According to these findings, we recommend several data resampling methods to enhance model performance. In our future work, we plan to 1) explore combining these methods, such as applying oversampling techniques both to the raw data and in the feature space, to enrich and broaden the dataset for more effective model training; and 2) investigate the role of loss functions in model performance. Although these functions do not alter the data distribution, they adjust weights, which can influence the overall effectiveness of log anomaly detection models. Our aim is to gain deeper insights and optimize existing approaches through thorough exploration in this domain.

References

- [1] M. Gegick, P. Rotella, and T. Xie, “Identifying security bug reports via text mining: An industrial case study,” in *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*. IEEE, 2010, pp. 11–20.
- [2] F. Peters, T. T. Tun, Y. Yu, and B. Nuseibeh, “Text filtering and ranking for security bug report prediction,” *IEEE Transactions on Software Engineering*, vol. 45, no. 6, pp. 615–631, 2017.
- [3] K. Goseva-Popstojanova and J. Tyo, “Identification of security related bug reports via text mining using supervised and unsupervised classification,” in *2018 IEEE International conference on software quality, reliability and security (QRS)*. IEEE, 2018, pp. 344–355.
- [4] R. Shu, T. Xia, L. Williams, and T. Menzies, “Better security bug report classification via hyperparameter optimization,” *arXiv preprint arXiv:1905.06872*, 2019.
- [5] Y. Jiang, P. Lu, X. Su, and T. Wang, “Ltrwes: A new framework for security bug report detection,” *Information and Software Technology*, vol. 124, p. 106314, 2020.
- [6] S. Chen, X. Xie, B. Yin, Y. Ji, L. Chen, and B. Xu, “Stay professional and efficient: automatically generate titles for your bug reports,” in *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2020, pp. 385–397.
- [7] T. Shi, Y. Keneshloo, N. Ramakrishnan, and C. K. Reddy, “Neural abstractive text summarization with sequence-to-sequence models,” *ACM Transactions on Data Science*, vol. 2, no. 1, pp. 1–37, 2021.
- [8] M. Du, F. Li, G. Zheng, and V. Srikumar, “Deeplog: Anomaly detection and diagnosis from system logs through deep learning,” in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2017, pp. 1285–1298.
- [9] W. Meng, Y. Liu, Y. Zhu, S. Zhang, D. Pei, Y. Liu, Y. Chen, R. Zhang, S. Tao, P. Sun *et al.*, “Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs,” in *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI)*, 2019, pp. 4739–4745.

- [10] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li *et al.*, “Robust log-based anomaly detection on unstable log data,” in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/SIGSOFT FSE)*, 2019, pp. 807–817.
- [11] V.-H. Le and H. Zhang, “Log-based anomaly detection without log parsing,” in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2021, pp. 492–504.
- [12] V. H. Le and H. Zhang, “Log-based anomaly detection with deep learning: How far are we?” in *Proceedings of the 44th international conference on software engineering*, 2022, pp. 1356–1367.
- [13] L. Yang, J. Chen, Z. Wang, W. Wang, J. Jiang, X. Dong, and W. Zhang, “Semi-supervised log-based anomaly detection via probabilistic label estimation,” in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 1448–1460.
- [14] C. Lee, T. Yang, Z. Chen, Y. Su, Y. Yang, and M. R. Lyu, “Heterogeneous anomaly detection for software systems via semi-supervised cross-modal attention,” in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, 2023.
- [15] S. Minaee, N. Kalchbrenner, E. Cambria, N. Nikzad, M. Chenaghlu, and J. Gao, “Deep learning-based text classification: a comprehensive review,” *ACM computing surveys (CSUR)*, vol. 54, no. 3, pp. 1–40, 2021.
- [16] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, “Indexing by latent semantic analysis,” *Journal of the American society for information science*, vol. 41, no. 6, pp. 391–407, 1990.
- [17] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, “Distributed representations of words and phrases and their compositionality,” *Advances in neural information processing systems*, vol. 26, 2013.
- [18] M. E. Peters, M. Neumann, L. Zettlemoyer, and W.-t. Yih, “Dissecting contextual word embeddings: Architecture and representation,” *arXiv preprint arXiv:1808.08949*, 2018.
- [19] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, “Improving language understanding with unsupervised learning,” 2018.
- [20] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” *arXiv preprint arXiv:1810.04805*, 2018.
- [21] X. Wu, W. Zheng, X. Chen, Y. Zhao, T. Yu, and D. Mu, “Improving high-impact bug report prediction with combination of interactive machine learning and active learning,” *Information and Software Technology*, vol. 133, p. 106530, 2021.

References

- [22] G. Klein, Y. Kim, Y. Deng, J. Senellart, and A. M. Rush, “Opennmt: Open-source toolkit for neural machine translation,” *arXiv preprint arXiv:1701.02810*, 2017.
- [23] A. Joulin, E. Grave, P. Bojanowski, M. Douze, H. Jégou, and T. Mikolov, “Fasttext. zip: Compressing text classification models,” *arXiv preprint arXiv:1612.03651*, 2016.
- [24] M. Bozorgi, L. K. Saul, S. Savage, and G. M. Voelker, “Beyond heuristics: learning to classify vulnerabilities and predict exploits,” in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2010, pp. 105–114.
- [25] S. Feng, J. Keung, X. Yu, Y. Xiao, and M. Zhang, “Investigation on the stability of smote-based oversampling techniques in software defect prediction,” *Information and Software Technology*, vol. 139, p. 106662, 2021.
- [26] X. Yu, J. Keung, Y. Xiao, S. Feng, F. Li, and H. Dai, “Predicting the precise number of software defects: Are we there yet?” *Information and Software Technology*, vol. 146, p. 106847, 2022.
- [27] X. Yu, J. Liu, J. W. Keung, Q. Li, K. E. Bennin, Z. Xu, J. Wang, and X. Cui, “Improving ranking-oriented defect prediction using a cost-sensitive ranking svm,” *IEEE Transactions on Reliability*, vol. 69, no. 1, pp. 139–153, 2019.
- [28] K. Zhao, Z. Xu, T. Zhang, Y. Tang, and M. Yan, “Simplified deep forest model based just-in-time defect prediction for android mobile apps,” *IEEE Transactions on Reliability*, vol. 70, no. 2, pp. 848–859, 2021.
- [29] X. Yu, K. E. Bennin, J. Liu, J. W. Keung, X. Yin, and Z. Xu, “An empirical study of learning to rank techniques for effort-aware defect prediction,” in *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2019, pp. 298–309.
- [30] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, “What makes a good bug report?” in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, 2008, pp. 308–318.
- [31] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” *Advances in neural information processing systems*, vol. 30, 2017.
- [32] C.-Y. Lin, “Rouge: A package for automatic evaluation of summaries,” in *Text summarization branches out*, 2004, pp. 74–81.
- [33] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, “Bleu: a method for automatic evaluation of machine translation,” in *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, 2002, pp. 311–318.

- [34] F. Kong, J. Li, B. Jiang, H. Wang, and H. Song, “Integrated generative model for industrial anomaly detection via bidirectional lstm and attention mechanism,” *IEEE Transactions on Industrial Informatics*, vol. 19, no. 1, pp. 541–550, 2021.
- [35] M. Cinque, C. Esposito, and A. Pecchia, “Security log analysis in critical industrial systems exploiting game theoretic feature selection and evidence combination,” *IEEE Transactions on Industrial Informatics*, vol. 16, no. 6, pp. 3871–3880, 2019.
- [36] D. Wu, Z. Jiang, X. Xie, X. Wei, W. Yu, and R. Li, “Lstm learning with bayesian and gaussian processing for anomaly detection in industrial iot,” *IEEE Transactions on Industrial Informatics*, vol. 16, no. 8, pp. 5244–5253, 2019.
- [37] A. A. Cook, G. Mısırlı, and Z. Fan, “Anomaly detection for iot time-series data: A survey,” *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 6481–6494, 2019.
- [38] S. He, P. He, Z. Chen, T. Yang, Y. Su, and M. R. Lyu, “A survey on automated log analysis for reliability engineering,” *ACM Computing Surveys*, vol. 54, no. 6, pp. 1–37, 2021.
- [39] Q. Lin, H. Zhang, J.-G. Lou, Y. Zhang, and X. Chen, “Log clustering based problem identification for online service systems,” in *Proceedings of the 38th International Conference on Software Engineering Companion*, 2016, pp. 102–111.
- [40] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, “Detecting large-scale system problems by mining console logs,” in *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles (SOSP)*, 2009, pp. 117–132.
- [41] M. Wurzenberger, F. Skopik, M. Landauer, P. Greitbauer, R. Fiedler, and W. Kastner, “Incremental clustering for semi-supervised anomaly detection applied on log data,” in *Proceedings of the 12th International Conference on Availability, Reliability and Security (ARES)*, 2017, pp. 1–6.
- [42] A. Oliner and J. Stearley, “What supercomputers say: A study of five system logs,” in *37th annual IEEE/IFIP international conference on dependable systems and networks (DSN’07)*. IEEE, 2007, pp. 575–584.
- [43] Q. Fu, J.-G. Lou, Q. Lin, R. Ding, D. Zhang, and T. Xie, “Contextual analysis of program logs for understanding system behaviors,” in *2013 10th Working Conference on Mining Software Repositories (MSR)*. IEEE, 2013, pp. 397–400.
- [44] Q. Fu, J. Zhu, W. Hu, J.-G. Lou, R. Ding, Q. Lin, D. Zhang, and T. Xie, “Where do developers log? an empirical study on logging practices in industry,” in *Companion Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 24–33.
- [45] H. Jiang, X. Li, Z. Yang, and J. Xuan, “What causes my test alarm? automatic cause analysis for test alarms in system and integration testing,” in *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 2017, pp. 712–723.

References

- [46] C. Zhi, J. Yin, S. Deng, M. Ye, M. Fu, and T. Xie, “An exploratory study of logging configuration practice in java,” in *2019 IEEE international conference on software maintenance and evolution (ICSME)*. IEEE, 2019, pp. 459–469.
- [47] S. Lu, X. Wei, Y. Li, and L. Wang, “Detecting anomaly in big data system logs using convolutional neural network,” in *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*. IEEE, 2018, pp. 151–158.
- [48] X. Ma, J. Keung, P. He, Y. Xiao, X. Yu, and Y. Li, “A semi-supervised approach for industrial anomaly detection via self-adaptive clustering,” *IEEE Transactions on Industrial Informatics*, 2023.
- [49] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, “Smote: synthetic minority over-sampling technique,” *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.
- [50] H. He, Y. Bai, E. A. Garcia, and S. Li, “Adasyn: Adaptive synthetic sampling approach for imbalanced learning,” in *2008 IEEE international joint conference on neural networks (IEEE world congress on computational intelligence)*. Ieee, 2008, pp. 1322–1328.
- [51] I. Mani and I. Zhang, “knn approach to unbalanced data distributions: a case study involving information extraction,” in *Proceedings of workshop on learning from imbalanced datasets*, vol. 126, no. 1. ICML, 2003, pp. 1–7.
- [52] M. R. Smith, T. Martinez, and C. Giraud-Carrier, “An instance level analysis of data complexity,” *Machine learning*, vol. 95, pp. 225–256, 2014.
- [53] G. E. Batista, R. C. Prati, and M. C. Monard, “A study of the behavior of several methods for balancing machine learning training data,” *ACM SIGKDD explorations newsletter*, vol. 6, no. 1, pp. 20–29, 2004.
- [54] N. Jalbert and W. Weimer, “Automated duplicate detection for bug tracking systems,” in *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*. IEEE, 2008, pp. 52–61.
- [55] C. Sun, D. Lo, X. Wang, J. Jiang, and S.-C. Khoo, “A discriminative model approach for accurate duplicate bug report retrieval,” in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*, 2010, pp. 45–54.
- [56] J. Deshmukh, K. Annervaz, S. Podder, S. Sengupta, and N. Dubash, “Towards accurate duplicate bug retrieval using deep learning techniques,” in *2017 IEEE International conference on software maintenance and evolution (ICSME)*. IEEE, 2017, pp. 115–124.

- [57] N. Ebrahimi, A. Trabelsi, M. S. Islam, A. Hamou-Lhadj, and K. Khanmohammadi, “An hmm-based approach for automatic detection and classification of duplicate bug reports,” *Information and Software Technology*, vol. 113, pp. 98–109, 2019.
- [58] B. S. Neysiani, S. M. Babamir, and M. Aritsugi, “Efficient feature extraction model for validation performance improvement of duplicate bug report detection in software bug triage systems,” *Information and Software Technology*, vol. 126, p. 106344, 2020.
- [59] Y. Tian, D. Lo, X. Xia, and C. Sun, “Automated prediction of bug report priority using multi-factor analysis,” *Empirical Software Engineering*, vol. 20, no. 5, pp. 1354–1383, 2015.
- [60] T. Zhang, G. Yang, B. Lee, and A. Chan, “Predicting severity of bug report by mining bug repository with concept profile,” *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, 2015.
- [61] Y. Tong and X. Zhang, “Crowdsourced test report prioritization considering bug severity,” *Information and Software Technology*, p. 106668, 2021.
- [62] X. Ye, R. Bunesu, and C. Liu, “Mapping bug reports to relevant files: A ranking model, a fine-grained benchmark, and feature evaluation,” *IEEE Transactions on Software Engineering*, vol. 42, no. 4, pp. 379–402, 2015.
- [63] A. N. Lam, A. T. Nguyen, H. A. Nguyen, and T. N. Nguyen, “Combining deep learning with information retrieval to localize buggy files for bug reports (n),” in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2015, pp. 476–481.
- [64] Z. Li, Z. Jiang, X. Chen, K. Cao, and Q. Gu, “Laprob: A label propagation-based software bug localization method,” *Information and Software Technology*, vol. 130, p. 106410, 2021.
- [65] R. Almhana, M. Kessentini, and W. Mkaouer, “Method-level bug localization using hybrid multi-objective search,” *Information and Software Technology*, vol. 131, p. 106474, 2021.
- [66] M. Kim and E. Lee, “Manq: Many-objective optimization-based automatic query reduction for ir-based bug localization,” *Information and Software Technology*, vol. 125, p. 106334, 2020.
- [67] M. Hamill and K. Goseva-Popstojanova, “Analyzing and predicting effort associated with finding and fixing software faults,” *Information and Software Technology*, vol. 87, pp. 1–18, 2017.
- [68] K. Goseva-Popstojanova and J. Tyo, “Experience report: security vulnerability profiles of mission critical software: empirical analysis of security related bug reports,” in *2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2017, pp. 152–163.

References

- [69] S. Panichella, G. Canfora, and A. Di Sorbo, ““won’t we fix this issue?” qualitative characterization and automated identification of wontfix issues on github,” *Information and Software Technology*, p. 106665, 2021.
- [70] K. Goseva-Popstojanova and A. Perhinschi, “On the capability of static code analysis to detect security vulnerabilities,” *Information and Software Technology*, vol. 68, pp. 18–33, 2015.
- [71] S. Kim, H. Zhang, R. Wu, and L. Gong, “Dealing with noise in defect prediction,” in *2011 33rd International Conference on Software Engineering (ICSE)*. IEEE, 2011, pp. 481–490.
- [72] R. Storn and K. Price, “Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces,” *Journal of global optimization*, vol. 11, no. 4, pp. 341–359, 1997.
- [73] W. Fu, T. Menzies, and X. Shen, “Tuning for software analytics: Is it really necessary?” *Information and Software Technology*, vol. 76, pp. 135–146, 2016.
- [74] C. Sun, D. Lo, S.-C. Khoo, and J. Jiang, “Towards more accurate retrieval of duplicate bug reports,” in *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*. IEEE, 2011, pp. 253–262.
- [75] D. R. Radev, E. Hovy, and K. McKeown, “Introduction to the special issue on summarization,” *Computational linguistics*, vol. 28, no. 4, pp. 399–408, 2002.
- [76] J. Cheng and M. Lapata, “Neural summarization by extracting sentences and words,” in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics, 2016.
- [77] R. Nallapati, F. Zhai, and B. Zhou, “Summarunner: A recurrent neural network based sequence model for extractive summarization of documents,” in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.
- [78] Q. Zhou, N. Yang, F. Wei, S. Huang, M. Zhou, and T. Zhao, “Neural document summarization by jointly learning to score and select sentences,” in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*. Association for Computational Linguistics, 2018, pp. 654–663.
- [79] S. Narayan, S. B. Cohen, and M. Lapata, “Ranking sentences for extractive summarization with reinforcement learning,” in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*. Association for Computational Linguistics, 2018, pp. 1747–1759.
- [80] X. Zhang, M. Lapata, F. Wei, and M. Zhou, “Neural latent extractive document summarization,” in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*. Association for Computational Linguistics, 2018, pp. 779–784.

- [81] Y. Dong, Y. Shen, E. Crawford, H. van Hoof, and J. C. K. Cheung, “Banditsum: Extractive summarization as a contextual bandit,” in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*. Association for Computational Linguistics, 2018, pp. 3739–3748.
- [82] Y. Liu, “Fine-tune bert for extractive summarization,” *arXiv preprint arXiv:1903.10318*, 2019.
- [83] 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)*. Association for Computational Linguistics, Jun. 2019, pp. 4171–4186.
- [84] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” in *Advances in neural information processing systems*, 2014, pp. 3104–3112.
- [85] K. Cho, B. van Merriënboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, “Learning phrase representations using RNN encoder-decoder for statistical machine translation,” in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*. ACL, 2014, pp. 1724–1734.
- [86] D. Bahdanau, K. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.
- [87] S. Shen, Y. Cheng, Z. He, W. He, H. Wu, M. Sun, and Y. Liu, “Minimum risk training for neural machine translation,” in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics, 2016.
- [88] D. Bahdanau, J. Chorowski, D. Serdyuk, P. Brakel, and Y. Bengio, “End-to-end attention-based large vocabulary speech recognition,” in *2016 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 2016, pp. 4945–4949.
- [89] Y. Miao, M. Gowayyed, and F. Metze, “Eesen: End-to-end speech recognition using deep rnn models and wfst-based decoding,” in *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*. IEEE, 2015, pp. 167–174.
- [90] S. Venugopalan, M. Rohrbach, J. Donahue, R. Mooney, T. Darrell, and K. Saenko, “Sequence to sequence-video to text,” in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 4534–4542.

References

- [91] A. M. Rush, S. Chopra, and J. Weston, “A neural attention model for abstractive sentence summarization,” in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*. The Association for Computational Linguistics, 2015, pp. 379–389.
- [92] R. Nallapati, B. Xiang, and B. Zhou, “Sequence-to-sequence rnns for text summarization,” 2016.
- [93] L. Dong, N. Yang, W. Wang, F. Wei, X. Liu, Y. Wang, J. Gao, M. Zhou, and H. Hon, “Unified language model pre-training for natural language understanding and generation,” in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada, 2019*, pp. 13 042–13 054.
- [94] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, “BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension,” in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*. Association for Computational Linguistics, 2020, pp. 7871–7880.
- [95] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, “Roberta: A robustly optimized BERT pretraining approach,” *CoRR*, vol. abs/1907.11692, 2019. [Online]. Available: <http://arxiv.org/abs/1907.11692>
- [96] J. Zhang, Y. Zhao, M. Saleh, and P. Liu, “Pegasus: Pre-training with extracted gap-sentences for abstractive summarization,” in *International Conference on Machine Learning*. PMLR, 2020, pp. 11 328–11 339.
- [97] Y. Liu and M. Lapata, “Text summarization with pretrained encoders,” 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, Hong Kong, China, November 3-7, 2019*. Association for Computational Linguistics, 2019, pp. 3728–3738.
- [98] Y. Liang, Y. Zhang, H. Xiong, and R. Sahoo, “Failure prediction in ibm bluegene/l event logs,” in *7th IEEE International Conference on Data Mining (ICDM)*. IEEE, 2007, pp. 583–588.
- [99] M. Chen, A. X. Zheng, J. Lloyd, M. I. Jordan, and E. Brewer, “Failure diagnosis using decision trees,” in *1st IEEE International Conference on Autonomic Computing (ICAC)*. IEEE, 2004, pp. 36–43.
- [100] M. Farshchi, J.-G. Schneider, I. Weber, and J. Grundy, “Experience report: Anomaly detection of cloud application operations using log and cloud metric correlation analysis,” in *2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2015, pp. 24–34.

- [101] K. Zhang, J. Xu, M. R. Min, G. Jiang, K. Pelechrinis, and H. Zhang, “Automated it system failure prediction: A deep learning approach,” in *2016 IEEE International Conference on Big Data (Big Data)*. IEEE, 2016, pp. 1291–1300.
- [102] R. Vinayakumar, K. Soman, and P. Poornachandran, “Long short-term memory based operation log anomaly detection,” in *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. IEEE, 2017, pp. 236–242.
- [103] X. Li, P. Chen, L. Jing, Z. He, and G. Yu, “Swisslog: Robust and unified deep learning based log anomaly detection for diverse faults,” in *31st IEEE International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2020, pp. 92–103.
- [104] C. Zhang, X. Peng, C. Sha, K. Zhang, Z. Fu, X. Wu, Q. Lin, and D. Zhang, “Deeptralog: Trace-log combined microservice anomaly detection through graph-based deep learning,” in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 623–634.
- [105] S. Hashemi and M. Mäntylä, “Sialog: detecting anomalies in software execution logs using the siamese network,” *Automated Software Engineering*, vol. 29, no. 2, p. 61, 2022.
- [106] Y. Fu, K. Liang, and J. Xu, “Mlog: Mogrifier lstm-based log anomaly detection approach using semantic representation,” *IEEE Transactions on Services Computing*, 2023.
- [107] S. Hashemi and M. Mäntylä, “Onelog: towards end-to-end software log anomaly detection,” *Automated Software Engineering*, vol. 31, no. 2, p. 37, 2024.
- [108] C. Zhang, T. Jia, G. Shen, P. Zhu, and Y. Li, “Metalog: Generalizable cross-system anomaly detection from logs with meta-learning,” in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 2024, pp. 1–12.
- [109] H. Guo, J. Yang, J. Liu, J. Bai, B. Wang, Z. Li, T. Zheng, B. Zhang, J. Peng, and Q. Tian, “Logformer: A pre-train and tuning pipeline for log anomaly detection,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 1, 2024, pp. 135–143.
- [110] X. Xie, S. Jian, C. Huang, F. Yu, and Y. Deng, “Logrep: Log-based anomaly detection by representing both semantic and numeric information in raw messages,” in *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2023, pp. 194–206.
- [111] S. He, Y. Lei, Y. Zhang, K. Xie, and P. K. Sharma, “Parameter-efficient log anomaly detection based on pre-training model and lora,” in *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2023, pp. 207–217.

References

- [112] W. Wang, S. Lu, J. Luo, and C. Wu, “Deepuserlog: Deep anomaly detection on user log using semantic analysis and key-value data,” in *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2023, pp. 172–182.
- [113] X. Wang, J. Song, X. Zhang, J. Tang, W. Gao, and Q. Lin, “Logonline: A semi-supervised log-based anomaly detector aided with online learning mechanism,” in *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2023, pp. 141–152.
- [114] T. Xiao, Z. Quan, Z.-J. Wang, Y. Le, Y. Du, X. Liao, K. Li, and K. Li, “Loader: A log anomaly detector based on transformer,” *IEEE Transactions on Services Computing*, 2023.
- [115] Y. Lee, J. Kim, and P. Kang, “Lanobert: System log anomaly detection based on bert masked language model,” *Applied Soft Computing*, vol. 146, p. 110689, 2023.
- [116] L. Yang, J. Chen, S. Gao, Z. Gong, H. Zhang, Y. Kang, and H. Li, “Try with simpler—an evaluation of improved principal component analysis in log-based anomaly detection,” *ACM Transactions on Software Engineering and Methodology*, 2023.
- [117] C. Seiffert, T. M. Khoshgoftaar, and J. Van Hulse, “Improving software-quality predictions with data sampling and boosting,” *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 39, no. 6, pp. 1283–1294, 2009.
- [118] W. Zheng, Y. Xun, X. Wu, Z. Deng, X. Chen, and Y. Sui, “A comparative study of class rebalancing methods for security bug report classification,” *IEEE Transactions on Reliability*, vol. 70, no. 4, pp. 1658–1670, 2021.
- [119] R. Shu, T. Xia, J. Chen, L. Williams, and T. Menzies, “How to better distinguish security bug reports (using dual hyperparameter optimization),” *Empirical Software Engineering*, vol. 26, pp. 1–37, 2021.
- [120] Y. Kamei, A. Monden, S. Matsumoto, T. Kakimoto, and K.-i. Matsumoto, “The effects of over and under sampling on fault-prone module detection,” in *First international symposium on empirical software engineering and measurement (ESEM 2007)*. IEEE, 2007, pp. 196–204.
- [121] L. Pelayo and S. Dick, “Evaluating stratification alternatives to improve software defect prediction,” *IEEE transactions on reliability*, vol. 61, no. 2, pp. 516–525, 2012.
- [122] M. Tan, L. Tan, S. Dara, and C. Mayeux, “Online defect prediction for imbalanced data,” in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 2. IEEE, 2015, pp. 99–108.

- [123] K. E. Bennin, J. Keung, A. Monden, P. Phannachitta, and S. Mensah, “The significant effects of data sampling approaches on software defect prioritization and classification,” in *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 2017, pp. 364–373.
- [124] S. Huda, K. Liu, M. Abdelrazek, A. Ibrahim, S. Alyahya, H. Al-Dossari, and S. Ahmad, “An ensemble oversampling model for class imbalance problem in software defect prediction,” *IEEE access*, vol. 6, pp. 24 184–24 195, 2018.
- [125] K. E. Bennin, J. W. Keung, and A. Monden, “On the relative value of data resampling approaches for software defect prediction,” *Empirical Software Engineering*, vol. 24, pp. 602–636, 2019.
- [126] H. Xu, R. Duan, S. Yang, and L. Guo, “An empirical study on data sampling for just-in-time defect prediction,” in *Artificial Intelligence and Security: 7th International Conference, ICAIS 2021, Dublin, Ireland, July 19–23, 2021, Proceedings, Part II 7*. Springer, 2021, pp. 54–69.
- [127] R. Yedida and T. Menzies, “On the value of oversampling for deep learning in software defect prediction,” *IEEE Transactions on Software Engineering*, vol. 48, no. 8, pp. 3103–3116, 2021.
- [128] F. Pecorelli, D. Di Nucci, C. De Roover, and A. De Lucia, “A large empirical assessment of the role of data balancing in machine-learning-based code smell detection,” *Journal of Systems and Software*, vol. 169, p. 110693, 2020.
- [129] F. Li, K. Zou, J. W. Keung, X. Yu, S. Feng, and Y. Xiao, “On the relative value of imbalanced learning for code smell detection,” *Software: Practice and Experience*, 2023.
- [130] K. E. Bennin, J. Keung, P. Phannachitta, A. Monden, and S. Mensah, “Mahakil: Diversity based oversampling approach to alleviate the class imbalance issue in software defect prediction,” *IEEE Transactions on Software Engineering*, vol. 44, no. 6, pp. 534–550, 2017.
- [131] N. Lunardon, G. Menardi, and N. Torelli, “Rose: a package for binary imbalanced learning.” *R journal*, vol. 6, no. 1, 2014.
- [132] Y. Heryadi and H. L. H. S. Warnars, “Learning temporal representation of transaction amount for fraudulent transaction recognition using cnn, stacked lstm, and cnn-lstm,” in *2017 IEEE International Conference on Cybernetics and Computational Intelligence (CyberneticsCom)*. IEEE, 2017, pp. 84–89.
- [133] A. Roy, J. Sun, R. Mahoney, L. Alonzi, S. Adams, and P. Beling, “Deep learning detecting fraud in credit card transactions,” in *2018 systems and information engineering design symposium (SIEDS)*. IEEE, 2018, pp. 129–134.
- [134] P. Raghavan and N. El Gayar, “Fraud detection using machine learning and deep learning,” in *2019 international conference on computational intelligence and knowledge economy (ICCIKE)*. IEEE, 2019, pp. 334–339.

References

- [135] W. Hilal, S. A. Gadsden, and J. Yawney, “Financial fraud: a review of anomaly detection techniques and recent advances,” *Expert systems With applications*, vol. 193, p. 116429, 2022.
- [136] K. M. Tsiouris, V. C. Pezoulas, M. Zervakis, S. Konitsiotis, D. D. Koutsouris, and D. I. Fotiadis, “A long short-term memory deep learning network for the prediction of epileptic seizures using eeg signals,” *Computers in biology and medicine*, vol. 99, pp. 24–37, 2018.
- [137] J. M. Johnson and T. M. Khoshgoftaar, “Survey on deep learning with class imbalance,” *Journal of Big Data*, vol. 6, no. 1, pp. 1–54, 2019.
- [138] F. Li, H. Tang, S. Shang, K. Mathiak, and F. Cong, “Classification of heart sounds using convolutional neural network,” *Applied Sciences*, vol. 10, no. 11, p. 3956, 2020.
- [139] T. Fernando, H. Gammulle, S. Denman, S. Sridharan, and C. Fookes, “Deep learning for medical anomaly detection—a survey,” *ACM Computing Surveys (CSUR)*, vol. 54, no. 7, pp. 1–37, 2021.
- [140] J. Yang, S. Li, Z. Wang, H. Dong, J. Wang, and S. Tang, “Using deep learning to detect defects in manufacturing: a comprehensive survey and current challenges,” *Materials*, vol. 13, no. 24, p. 5755, 2020.
- [141] A. Abdelkhalek and M. Mashaly, “Addressing the class imbalance problem in network intrusion detection systems using data resampling and deep learning,” *The Journal of Supercomputing*, pp. 1–34, 2023.
- [142] X.-w. Chen and M. Wasikowski, “Fast: a roc-based feature selection metric for small samples and imbalanced data classification problems,” in *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2008, pp. 124–132.
- [143] X. Ma, J. Keung, Z. Yang, X. Yu, Y. Li, and H. Zhang, “Casms: Combining clustering with attention semantic model for identifying security bug reports,” *Information and Software Technology*, vol. 147, p. 106906, 2022.
- [144] J. Lilleberg, Y. Zhu, and Y. Zhang, “Support vector machines and word2vec for text classification with semantic features,” in *2015 IEEE 14th International Conference on Cognitive Informatics & Cognitive Computing (ICCI* CC)*. IEEE, 2015, pp. 136–140.
- [145] J. Ramos *et al.*, “Using tf-idf to determine word relevance in document queries,” in *Proceedings of the first instructional conference on machine learning*, vol. 242, no. 1. Citeseer, 2003, pp. 29–48.
- [146] A. Aizawa, “An information-theoretic perspective of tf-idf measures,” *Information Processing & Management*, vol. 39, no. 1, pp. 45–65, 2003.
- [147] D. Wijayasekara, M. Manic, J. L. Wright, and M. McQueen, “Mining bug databases for unidentified software vulnerabilities,” in *2012 5th International Conference on Human System Interactions*. IEEE, 2012, pp. 89–96.

- [148] M. Esposito, E. Damiano, A. Minutolo, G. De Pietro, and H. Fujita, “Hybrid query expansion using lexical resources and word embeddings for sentence retrieval in question answering,” *Information Sciences*, vol. 514, pp. 88–105, 2020.
- [149] R. A. Stein, P. A. Jaques, and J. F. Valiati, “An analysis of hierarchical text classification using word embeddings,” *Information Sciences*, vol. 471, pp. 216–232, 2019.
- [150] V. Di Carlo, F. Bianchi, and M. Palmonari, “Training temporal word embeddings with a compass,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 6326–6334.
- [151] N. Garg, L. Schiebinger, D. Jurafsky, and J. Zou, “Word embeddings quantify 100 years of gender and ethnic stereotypes,” *Proceedings of the National Academy of Sciences*, vol. 115, no. 16, pp. E3635–E3644, 2018.
- [152] V. Tshitoyan, J. Dagdelen, L. Weston, A. Dunn, Z. Rong, O. Kononova, K. A. Persson, G. Ceder, and A. Jain, “Unsupervised word embeddings capture latent knowledge from materials science literature,” *Nature*, vol. 571, no. 7763, pp. 95–98, 2019.
- [153] X. Dai, M. Bikdash, and B. Meyer, “From social media to public health surveillance: Word embedding based clustering method for twitter classification,” in *SoutheastCon 2017*. IEEE, 2017, pp. 1–7.
- [154] L. Xiao, G. Wang, and Y. Zuo, “Research on patent text classification based on word2vec and lstm,” in *2018 11th International Symposium on Computational Intelligence and Design (ISCID)*, vol. 1. IEEE, 2018, pp. 71–74.
- [155] J. Gao, Y. He, X. Zhang, and Y. Xia, “Duplicate short text detection based on word2vec,” in *2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS)*. IEEE, 2017, pp. 33–37.
- [156] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” *arXiv preprint arXiv:1301.3781*, 2013.
- [157] X. Rong, “word2vec parameter learning explained,” *arXiv preprint arXiv:1411.2738*, 2014.
- [158] C. Boutsidis, P. Drineas, and M. W. Mahoney, “Unsupervised feature selection for the k -means clustering problem,” in *Advances in Neural Information Processing Systems*, 2009, pp. 153–161.
- [159] S. Alelyani, J. Tang, and H. Liu, “Feature selection for clustering: A review,” *Data Clustering*, pp. 29–60, 2018.
- [160] S. Xiang, F. Nie, and C. Zhang, “Learning a mahalanobis distance metric for data clustering and classification,” *Pattern recognition*, vol. 41, no. 12, pp. 3600–3612, 2008.

References

- [161] T. Li, S. Ma, and M. Ogihara, “Entropy-based criterion in categorical clustering,” in *Proceedings of the twenty-first international conference on Machine learning*, 2004, p. 68.
- [162] F. Murtagh and P. Contreras, “Methods of hierarchical clustering,” *arXiv preprint arXiv:1105.0121*, 2011.
- [163] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to data mining*. Pearson Education India, 2016.
- [164] G. Seif, “The 5 clustering algorithms data scientists need to know,” *Towards Data Science*, 2018.
- [165] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, “A density-based algorithm for discovering clusters in large spatial databases with noise.” in *Kdd*, vol. 96, no. 34, 1996, pp. 226–231.
- [166] K. Khan, S. U. Rehman, K. Aziz, S. Fong, and S. Sarasvady, “DbSCAN: Past, present and future,” in *The fifth international conference on the applications of digital information and web technologies (ICADIWT 2014)*. IEEE, 2014, pp. 232–238.
- [167] X. Xu, M. Ester, H.-P. Kriegel, and J. Sander, “A distribution-based clustering algorithm for mining in large spatial databases,” in *Proceedings 14th International Conference on Data Engineering*. IEEE, 1998, pp. 324–331.
- [168] M. Huang and F. Bian, “A grid and density based fast spatial clustering algorithm,” in *2009 International Conference on Artificial Intelligence and Computational Intelligence*, vol. 4. IEEE, 2009, pp. 260–263.
- [169] J. P. Gleeson, S. Melnik, and A. Hackett, “How clustering affects the bond percolation threshold in complex networks,” *Physical Review E*, vol. 81, no. 6, p. 066114, 2010.
- [170] A. Dharmarajan and T. Velmurugan, “Applications of partition based clustering algorithms: A survey,” in *2013 IEEE International Conference on Computational Intelligence and Computing Research*. IEEE, 2013, pp. 1–5.
- [171] D. Xu and Y. Tian, “A comprehensive survey of clustering algorithms,” *Annals of Data Science*, vol. 2, no. 2, pp. 165–193, 2015.
- [172] F. Liu and Y. Deng, “Determine the number of unknown targets in open world based on elbow method,” *IEEE Transactions on Fuzzy Systems*, vol. 29, no. 5, pp. 986–995, 2020.
- [173] T. M. Kodinariya and P. R. Makwana, “Review on determining number of cluster in k-means clustering,” *International Journal*, vol. 1, no. 6, pp. 90–95, 2013.
- [174] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu, “An efficient k-means clustering algorithm: Analysis and implementation,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 24, no. 7, pp. 881–892, 2002.

- [175] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, “Benchmarking classification models for software defect prediction: A proposed framework and novel findings,” *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 485–496, 2008.
- [176] Z. Harry, “The optimality of naive bayes,” in *FLAIRS2004 conference*, 2004.
- [177] W. Afzal, “Using faults-slip-through metric as a predictor of fault-proneness,” in *2010 Asia Pacific Software Engineering Conference*. IEEE, 2010, pp. 414–422.
- [178] E. J. Weyuker, T. J. Ostrand, and R. M. Bell, “Do too many cooks spoil the broth? using the number of developers to enhance defect prediction models,” *Empirical Software Engineering*, vol. 13, no. 5, pp. 539–559, 2008.
- [179] J. G. Shanahan and N. Roma, “Improving svm text classification performance through threshold adjustment,” in *European Conference on Machine Learning*. Springer, 2003, pp. 361–372.
- [180] L. Breiman, “Random forests,” *Machine learning*, vol. 45, pp. 5–32, 2001.
- [181] C. M. Bishop *et al.*, *Neural networks for pattern recognition*. Oxford university press, 1995.
- [182] T. Cover and P. Hart, “Nearest neighbor pattern classification,” *IEEE transactions on information theory*, vol. 13, no. 1, pp. 21–27, 1967.
- [183] C. Zhou, C. Sun, Z. Liu, and F. Lau, “A c-lstm neural network for text classification,” *arXiv preprint arXiv:1511.08630*, 2015.
- [184] J. L. Elman, “Finding structure in time,” *Cognitive science*, vol. 14, no. 2, pp. 179–211, 1990.
- [185] J. Gao, J.-Y. Nie, G. Wu, and G. Cao, “Dependence language model for information retrieval,” in *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, 2004, pp. 170–177.
- [186] A. Graves, S. Fernández, and J. Schmidhuber, “Bidirectional lstm networks for improved phoneme classification and recognition,” in *International conference on artificial neural networks*. Springer, 2005, pp. 799–804.
- [187] A. Graves, N. Jaitly, and A.-r. Mohamed, “Hybrid speech recognition with deep bidirectional lstm,” in *2013 IEEE workshop on automatic speech recognition and understanding*. IEEE, 2013, pp. 273–278.
- [188] N. S. Madiraju, S. M. Sadat, D. Fisher, and H. Karimabadi, “Deep temporal clustering: Fully unsupervised learning of time-domain features,” *arXiv preprint arXiv:1802.01059*, 2018.
- [189] Y. Wu and W. Li, “Automatic audio chord recognition with midi-trained deep feature and blstm-crf sequence decoding model,” *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 27, no. 2, pp. 355–366, 2018.

References

- [190] B. Liu and S. Li, “Protdet-cch: protein remote homology detection by combining long short-term memory and ranking methods,” *IEEE/ACM transactions on computational biology and bioinformatics*, vol. 16, no. 4, pp. 1203–1210, 2018.
- [191] D. Quang and X. Xie, “Danq: a hybrid convolutional and recurrent deep neural network for quantifying the function of dna sequences,” *Nucleic acids research*, vol. 44, no. 11, pp. e107–e107, 2016.
- [192] D. Bahdanau, K. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” *arXiv preprint arXiv:1409.0473*, 2014.
- [193] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, “Show, attend and tell: Neural image caption generation with visual attention,” in *International conference on machine learning*. PMLR, 2015, pp. 2048–2057.
- [194] J. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio, “Attention-based models for speech recognition,” *arXiv preprint arXiv:1506.07503*, 2015.
- [195] U. Khandelwal, H. He, P. Qi, and D. Jurafsky, “Sharp nearby, fuzzy far away: How neural language models use context,” *arXiv preprint arXiv:1805.04623*, 2018.
- [196] S. Xingjian, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo, “Convolutional lstm network: A machine learning approach for precipitation nowcasting,” in *Advances in neural information processing systems*, 2015, pp. 802–810.
- [197] M. Ohira, Y. Kashiwa, Y. Yamatani, H. Yoshiyuki, Y. Maeda, N. Limsettho, K. Fujino, H. Hata, A. Ihara, and K. Matsumoto, “A dataset of high impact bugs: Manually-classified issue reports,” in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 2015, pp. 518–521.
- [198] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, “Tensorflow: A system for large-scale machine learning,” in *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, 2016, pp. 265–283.
- [199] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, “Scikit-learn: Machine learning in python,” *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.
- [200] Y. Jiang, B. Cukic, and Y. Ma, “Techniques for evaluating fault prediction models,” *Empirical Software Engineering*, vol. 13, no. 5, pp. 561–595, 2008.
- [201] J. M. Bland and D. G. Altman, “Statistics notes: measurement error,” *Bmj*, vol. 312, no. 7047, p. 1654, 1996.

- [202] X. Ma, J. W. Keung, X. Yu, H. Zou, J. Zhang, and Y. Li, “Attsum: A deep attention-based summarization model for bug report title generation,” *IEEE Transactions on Reliability*, 2023.
- [203] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le, “Xlnet: Generalized autoregressive pretraining for language understanding,” *Advances in neural information processing systems*, vol. 32, 2019.
- [204] M. Joshi, D. Chen, Y. Liu, D. S. Weld, L. Zettlemoyer, and O. Levy, “Spanbert: Improving pre-training by representing and predicting spans,” *Transactions of the Association for Computational Linguistics*, vol. 8, pp. 64–77, 2020.
- [205] J. Gu, Z. Lu, H. Li, and V. O. K. Li, “Incorporating copying mechanism in sequence-to-sequence learning,” in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics, 2016.
- [206] Y. Zhen, J. W. KEUNG, Y. Xiao, X. Yan, J. Zhi, and J. ZHANG, “On the significance of category prediction for code-comment synchronization,” *ACM Transactions on Software Engineering and Methodology*, 2022.
- [207] F. Zhang, X. Yu, J. Keung, F. Li, Z. Xie, Z. Yang, C. Ma, and Z. Zhang, “Improving stack overflow question title generation with copying enhanced codebert model and bi-modal information,” *Information and Software Technology*, vol. 148, p. 106922, 2022.
- [208] A. See, P. J. Liu, and C. D. Manning, “Get to the point: Summarization with pointer-generator networks,” in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, R. Barzilay and M. Kan, Eds. Association for Computational Linguistics, 2017, pp. 1073–1083.
- [209] Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler, “Aligning books and movies: Towards story-like visual explanations by watching movies and reading books,” in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 19–27.
- [210] S. Nagel, “Cc-news,” URL: <http://web.archive.org/save/http://commoncrawl.org/2016/10/newsdatasetavailable>, 2016.
- [211] A. Gokaslan and V. Cohen, “Openwebtext corpus,” URL: <https://skylion007.github.io/OpenWebTextCorpus>, 2019.
- [212] T. H. Trinh and Q. V. Le, “A simple method for commonsense reasoning,” *CoRR*, vol. abs/1806.02847, 2018. [Online]. Available: <http://arxiv.org/abs/1806.02847>
- [213] Z. Gao, X. Xia, J. Grundy, D. Lo, and Y.-F. Li, “Generating question titles for stack overflow from mined code snippets,” *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 29, no. 4, pp. 1–37, 2020.

References

- [214] S. Bird, “NLTK: the natural language toolkit,” in *ACL 2006, 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, Sydney, Australia, 17-21 July 2006*, N. Calzolari, C. Cardie, and P. Isabelle, Eds. The Association for Computer Linguistics, 2006. [Online]. Available: <https://aclanthology.org/P06-4018/>
- [215] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, “Language models are unsupervised multitask learners,” *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.
- [216] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015.
- [217] C.-Y. Lin and F. J. Och, “Orange: a method for evaluating automatic evaluation metrics for machine translation,” in *COLING 2004: Proceedings of the 20th International Conference on Computational Linguistics*, 2004, pp. 501–507.
- [218] Z. Yang, J. Keung, X. Yu, X. Gu, Z. Wei, X. Ma, and M. Zhang, “A multi-modal transformer-based code summarization approach for smart contracts,” in *2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC)*. IEEE, 2021, pp. 1–12.
- [219] G. M. Sullivan and R. Feinn, “Using effect size—or why the p value is not enough,” *Journal of graduate medical education*, vol. 4, no. 3, pp. 279–282, 2012.
- [220] M. R. Rhea, “Determining the magnitude of treatment effects in strength training research through the use of the effect size,” *The Journal of Strength & Conditioning Research*, vol. 18, no. 4, pp. 918–920, 2004.
- [221] J. Cohen, *Statistical power analysis for the behavioral sciences*. Routledge, 2013.
- [222] X. Hu, Q. Chen, H. Wang, X. Xia, D. Lo, and T. Zimmermann, “Correlating automated and human evaluation of code documentation generation quality,” *ACM Transactions on Software Engineering and Methodology*, 2021.
- [223] D. E. Hinkle, W. Wiersma, and S. G. Jurs, *Applied statistics for the behavioral sciences*. Houghton Mifflin College Division, 2003, vol. 663.
- [224] H. He and E. A. Garcia, “Learning from imbalanced data,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 9, pp. 1263–1284, 2009.
- [225] A. Raganato and J. Tiedemann, “An analysis of encoder representations in transformer-based machine translation,” in *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*. The Association for Computational Linguistics, 2018.

- [226] Y. Liu and M. Lapata, “Text summarization with pretrained encoders,” 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, pp. 3728–3738.
- [227] I. Loshchilov and F. Hutter, “Decoupled weight decay regularization,” in *7th International Conference on Learning Representations (ICLR), Poster*, 2019.
- [228] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, “Drain: An online log parsing approach with fixed depth tree,” in *2017 IEEE international conference on web services (ICWS)*. IEEE, 2017, pp. 33–40.
- [229] G. Salton and C. Buckley, “Term-weighting approaches in automatic text retrieval,” *Information processing & management*, vol. 24, no. 5, pp. 513–523, 1988.
- [230] S. Chakraborty, R. Krishna, Y. Ding, and B. Ray, “Deep learning based vulnerability detection: Are we there yet,” *IEEE Transactions on Software Engineering*, 2021.
- [231] X. Yang, S. Wang, Y. Li, and S. Wang, “Does data sampling improve deep learning-based vulnerability detection? yeas! and nays!” in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 2287–2298.
- [232] R. Malhotra and S. Kamal, “An empirical study to investigate oversampling methods for improving software defect prediction using imbalanced data,” *Neurocomputing*, vol. 343, pp. 120–140, 2019.
- [233] D. L. Wilson, “Asymptotic properties of nearest neighbor rules using edited data,” *IEEE Transactions on Systems, Man, and Cybernetics*, no. 3, pp. 408–421, 1972.
- [234] I. Tomek, “Two modifications of cnn.” 1976.
- [235] S. He, J. Zhu, P. He, and M. R. Lyu, “Loghub: A large collection of system log datasets towards automated log analytics,” *arXiv preprint arXiv:2008.06448*, 2020.
- [236] X. Liu, W. Liu, X. Di, J. Li, B. Cai, W. Ren, and H. Yang, “Lognads: Network anomaly detection scheme based on log semantics representation,” *Future Generation Computer Systems*, vol. 124, pp. 390–405, 2021.
- [237] Q. Song, Y. Guo, and M. Shepperd, “A comprehensive investigation of the role of imbalanced learning for software defect prediction,” *IEEE Transactions on Software Engineering*, vol. 45, no. 12, pp. 1253–1269, 2018.
- [238] J. Yao and M. Shepperd, “Assessing software defection prediction performance: Why using the matthews correlation coefficient matters,” in *Proceedings of the 24th International Conference on Evaluation and Assessment in Software Engineering*, 2020, pp. 120–129.

References

- [239] R. Moussa and F. Sarro, “On the use of evaluation measures for defect prediction studies,” in *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2022, pp. 101–113.
- [240] K. E. Bennin, A. Tahir, S. G. MacDonell, and J. Börstler, “An empirical study on the effectiveness of data resampling approaches for cross-project software defect prediction,” *IET Software*, vol. 16, no. 2, pp. 185–199, 2022.
- [241] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, “An empirical comparison of model validation techniques for defect prediction models,” *IEEE Transactions on Software Engineering*, vol. 43, no. 1, pp. 1–18, 2016.
- [242] G. Lemaître, F. Nogueira, and C. K. Aridas, “Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning,” *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 559–563, 2017.
- [243] M. T. Ribeiro, S. Singh, and C. Guestrin, ““why should i trust you?” explaining the predictions of any classifier,” in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, pp. 1135–1144.
- [244] C. Pornprasit, C. Tantithamthavorn, J. Jiarapakdee, M. Fu, and P. Thongtanunam, “Pyexplainer: Explaining the predictions of just-in-time defect models,” in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2021, pp. 407–418.
- [245] C. K. Tantithamthavorn and J. Jiarapakdee, “Explainable ai for software engineering,” in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2021, pp. 1–2.
- [246] B. Ledel and S. Herbold, “Studying the explanations for the automated prediction of bug and non-bug issues using lime and shap,” *arXiv preprint arXiv:2209.07623*, 2022.
- [247] M. Fan, W. Wei, X. Xie, Y. Liu, X. Guan, and T. Liu, “Can we trust your explanations? sanity checks for interpreters in android malware analysis,” *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 838–853, 2020.
- [248] J. Feichtner and S. Gruber, “Understanding privacy awareness in android app descriptions using deep learning,” in *Proceedings of the tenth ACM conference on data and application security and privacy*, 2020, pp. 203–214.
- [249] A. Warnecke, D. Arp, C. Wressnegger, and K. Rieck, “Evaluating explanation methods for deep learning in security,” in *2020 IEEE european symposium on security and privacy (EuroS&P)*. IEEE, 2020, pp. 158–174.
- [250] S. Lu, M. Wang, D. Wang, X. Wei, S. Xiao, Z. Wang, N. Han, and L. Wang, “Black-box attacks against log anomaly detection with adversarial examples,” *Information Sciences*, vol. 619, pp. 249–262, 2023.

- [251] A. Goodge, B. Hooi, S.-K. Ng, and W. S. Ng, “Robustness of autoencoders for anomaly detection under adversarial impact.” in *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI)*, 2020, pp. 1244–1250.
- [252] F. Liu, H. Liu, and W. Jiang, “Practical adversarial attacks on spatiotemporal traffic forecasting models,” in *Proceedings of the 36th Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2022, pp. 1–18.