SUPPORTING SOFTWARE ENGINEERS WITH LARGE LANGUAGE MODEL-BASED AUTOMATION

ZHANG, TING

SINGAPORE MANAGEMENT UNIVERSITY

2023

SUPPORTING SOFTWARE ENGINEERS WITH
LARGE LANGUAGE MODEL-BASED AUTOMATION

ZHANG, TING

Submitted to School of Computing and Information Systems
in partial fulfillment of the requirements for the Degree of

DOCTOR OF PHILOSOPHY

in Computer Science

Dissertation Committee

David Lo (Supervisor / Chair)
Professor
Singapore Management University

Lingxiao Jiang (Co-supervisor)
Associate Professor
Singapore Management University

Jing Jiang
Professor
Singapore Management University

Alexander Serebrenik
Professor
Eindhoven University of Technology

I hereby declare that this Ph.D. dissertation is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in this dissertation.

This dissertation has also not been submitted for any degree in any university previously.

Date: 31 October 2023
Signature:

ABSTRACT OF THE DISSERTATION

SUPPORTING SOFTWARE ENGINEERS WITH
LARGE LANGUAGE MODEL-BASED AUTOMATION

By

ZHANG, TING

In recent years, software engineering (SE) has witnessed significant growth, leading to the creation and sharing of an abundance of software artifacts such as source code, bug reports, and pull requests. Analyzing these artifacts is crucial for comprehending the sentiments of software developers and automating various SE tasks, ultimately leading to more human-centered automated SE and enhancing software development efficiency. However, the diverse and unstructured nature of software text poses a significant challenge to this analysis. In response, researchers have investigated a variety of approaches, including the utilization of natural language processing techniques. The advent of large language models (LLMs), ranging from smaller-size LLMs (sLLMs) like BERT to bigger ones (bLLMs) such as LLaMA, has ignited a growing interest in their potential for analyzing software-related text.

This dissertation explores how LLMs can automate different SE tasks involving classification, ranking, and generation tasks. In the first study, we assess the efficacy of sLLMs, such as BERT, in SE sentiment analysis, comparing them to existing SE-specific tools. Furthermore, we compare the performance of bLLMs with sLLMs in this context. In the second study, we address the issue of retrieving duplicate bug reports. First, we create a benchmark and then use bLLMs to enhance the accuracy of this process, with a specific focus on employing GPT-3.5 for suggesting duplicate bug reports. In the third study, we propose to leverage sLLMs to create precise and concise pull request titles.

In conclusion, this dissertation contributes to the SE field by exploring the potential of LLMs to support software developers in understanding sentiments and improving the efficiency of software development.

TABLE OF CONTENTS

Page

ACKNOWLEDGMENTS

I am profoundly grateful for the unwavering support and guidance I have received from an array of individuals who have been instrumental in my academic journey, research pursuits, and personal growth. Completing my PhD would not have been possible without their invaluable contributions.

First and foremost, I wish to convey my deep gratitude to my primary supervisor, Professor David Lo. His invaluable mentorship has undeniably served as the cornerstone of my academic achievements. In addition to offering me essential academic guidance, he has instilled within me an enduring passion for research and an unwavering dedication to the pursuit of excellence. Throughout my entire journey toward a Ph.D., his constant support and guidance have remained an abundant source of inspiration. I would like to wholeheartedly thank David for his steadfast support in facilitating my participation in many academic conferences. This not only allowed me to interact with senior scholars and fellow peers in our field but also played a pivotal role in fostering my growth as a researcher. Furthermore, I have had the privilege of learning from David the virtues of patience and kindness, qualities that I deeply admire and aspire to emulate.

I am deeply grateful for the vital contributions made by my dissertation committee members, including Associate Professor Lingxiao Jiang (my co-supervisor), Professor Jing Jiang, and Professor Alexander Serebrenik. I would like to express my heartfelt appreciation for the time they dedicated to my research. Their profound expertise and insightful perspectives pushed me to explore the frontiers of my study. Their constructive feedback and valuable suggestions played an instrumental role in enhancing both the quality and depth of my work, and I am truly thankful for their unwavering commitment to my academic growth.

I am deeply grateful to SEOW Pei Huan, Caroline Tan, BOO Chui Ngoh, Nikki Chew, Louise LOW, and Professor Baihua Zheng from the SMU SCIS post-graduate office. Their unwavering dedication in aiding me with various administrative tasks, ranging from logistical support to paperwork, has significantly alleviated the bureaucratic challenges that can often accompany higher education. Their invaluable behind-the-scenes assistance has allowed me to concentrate on my research and academic endeavors with a profound sense of ease. They efficiently managed every aspect, and for that, I am immensely appreciative. Thank you so much!

I would also like to extend my gratitude to SMU. I appreciate the opportunity to study at this esteemed institution. My journey at SMU has been profoundly meaningful, and it holds a special place in my heart. This has been a place where I have not only gained knowledge but also created cherished memories. Being a member of the SMU Aquathlon team has been a remarkable experience. I acquired the skill of front crawl thanks to our dedicated coach and the camaraderie of my teammates. To all of them, I extend my heartfelt thanks – you are the best! I am also grateful for the opportunity to represent our

club in cycling races. It has been an honor and a source of immense pride. Reflecting on these four years, I am filled with gratitude and contentment. Luckily, my journey at SMU will continue as I take on the role of a post-doctoral researcher in the coming new year.

The PhD journey is a collaborative endeavor, and I consider myself fortunate to have had the privilege of working alongside a group of exceptionally talented individuals. I want to extend my heartfelt gratitude to my collaborators and our SOAR lab mates, listed here in the order of when I had the pleasure of meeting them: Bowen Xu, Ferdian Thung, Hong Jin Kang, Gede Artha Azriadi Prana, Thong Hoang, Stefanus Agus Haryono, Muhammad Hilmi ASYROFI, Xin Zhou, Zhou Yang, Chengran Yang, Ratnadira Widyasari, Imam Nur Bani Yusuf, DongGyun Han, Venkatesh Vinayakarao, Divya Prabha Chandrasekaran, Jieke Shi, Junda He, Le-Cong Thanh, Truong Giang Nguyen, Kien Luong, Ivana Clairine Irsan, Kisub Kim, Chen Gong, Jiakun Liu, Zhipeng Zhao, Changan Niu, Pattarakrit Rattanukul, Yunbo Lyu, Chenyu Wang, Abir Bouraffa, Yue Liu, Shidong Pan and Zeshi Li. Together, we have shared numerous moments of discovery, celebration, and camaraderie, all of which have not only enriched my research but also significantly impacted my personal and professional life. I am particularly grateful to have forged meaningful friendships with some of you. Additionally, I wish to thank Prof. Xuan-Bach B. Le for providing invaluable guidance during my search for faculty positions. The SMU SOAR lab has become a second family to me, and I am immensely proud and grateful to be a part of it.

I want to convey my sincere appreciation to my supervisor at Sun Yat-sen University, Associate Professor Hankui Zhuo, for generously dedicating his valuable time to craft a recommendation letter supporting my post-graduate study applications. I am also deeply grateful to Professor Hankui Zhuo for welcoming me into the XPlan lab at SYSU, marking the inception of my research journey. My gratitude extends to Professor Di Wu, who also devoted time to composing recommendation letters for me. Furthermore, I wish to express my profound thanks to my esteemed professors and dear friends at SYSU, Chuanji Hu and Jianjun Tong. Not only have you been exceptional friends, but you have also been a continual source of inspiration and guidance in my life's journey. I also wish to acknowledge the members of my major-switching interview committee. I am appreciative of the opportunity you provided. Without your acceptance, I would not be the person I am today. Thank you for believing in me.

I would like to express my heartfelt gratitude to my YouTube audience. Your support has been instrumental in achieving a remarkable milestone of over 1 million views across all my videos. Your kind words, compliments, and stories of how my content has led to satisfying job opportunities have filled me with deep appreciation. I extend my heartfelt thanks to those who have also provided financial support. I am grateful, and a profound sense of accomplishment washes over me. As a content creator, I take immense pride in the knowledge that I have had a positive impact on the lives of many. While I may not become a hero, knowing that my small contributions can make our world a better place is immensely gratifying. Despite my limited activity this year, I assure you that I will return with even more content next year after my graduation. Thank you for being part of this incredible journey.

The work in this dissertation was published in three different venues: ICSME 2020 (Chapter 2), TOSEM (Chapter 3), and ICSME 2022 (Chapter 4). I thank the anonymous reviewers who reviewed the submissions and provided insightful comments to improve the work. I would also like to thank the audience who attended and commented on the presentations of the work in this dissertation.

Last but not least, I want to give my sincere thanks to my beloved family and friends. It is impossible for me to ask for more from any of you. You are, without a doubt, the absolute best. The unwavering support of my family, particularly my parents, Cuifang Shen and Dengfeng Zhang, has been my anchor. Their encouragement, love, and belief in my abilities have been the driving force behind my academic endeavors. Their sacrifices and unwavering support throughout the years testify to the power of family bonds in nurturing one's ambitions. I also wish to acknowledge my grandparents, Zhengyuan Zhang, Guiping Zhu, Cuilian Chen (who passed away in 2017), and Yugen Shen (who passed away in 2009). I dearly wish for more time to have shared with you. Additionally, I wish to extend my sincere thanks to my uncle Feng Shen and my aunt Chuimei Gu (who is courageously battling cancer now). I genuinely wish you a life filled with good health and happiness. My love for my family knows no bounds, and I am profoundly thankful for your unwavering support. I extend my heartfelt thanks to my best friend, Yi Wu, for being constant in my life. Your friendship has been a source of solace, laughter, and shared experiences that have made this arduous journey a bit easier. Your presence has provided much-needed balance in my life. I also want to thank all my friends, including but not limited to Yingying Zeng, Xing Li, Zilin Luo, Ying Luo, and Tianci Tang. Sorry, I cannot exhaustly mention all the names here, but I keep your names in my heart. I am grateful for my partner, Zhaozheng Chen, whose unwavering support, understanding, and shared dreams have provided me with a sense of purpose and motivation. I am confident that, together, we will continue to create a brighter future, and I eagerly anticipate the remarkable journey that lies ahead.

To all the people who love me and whom I love.

Chapter 1

Introduction

In recent years, software engineering (SE) has seen significant growth and innovation due to the increasing use of software in nearly every aspect of modern life. This has put SE at the forefront of technological progress.

Despite advancements, software development remains a labor-intensive task that relies heavily on human input. This has created a need for automated solutions to reduce software development and maintenance workload.

To address these issues, this dissertation focuses on using Large Language Models (LLMs) to improve automated SE. Automated SE aims to make the software development process more efficient and produce higher-quality software.

This dissertation encompasses three studies, each dedicated to classification, ranking, and text generation tasks respectively:

1. Sentiment Analysis for Software Engineering (SA4SE): This study seeks to discern sentiment in software-related textual data, ranging from APP reviews on Google Play to pull request feedback on GitHub and comments on Stack Overflow. The

end goal is to classify these text units into negative, positive, or neutral sentiment categories.

2. Duplicate Bug Report Detection (DBRD): In this study, we tackle a ranking task. Given a newly submitted bug report (BR), we aim to recommend a list of existing BRs that the submitted report might be duplicating.

3. Automatic Pull Request (PR) Title Generation: Falling under text generation, this segment focuses on crafting PR titles. The generation process leverages the PR description, associated commit messages, and linked issue titles to ensure relevancy and coherence.

Throughout these three studies, we leverage the capabilities of LLMs to expedite and optimize the entire process. LLMs refer to Transformer models that are pre-trained on massive text data. In recent years, LLMs have emerged as game-changing methods in artificial intelligence and natural language processing [189, 36, 190]. These LLMs come in two flavors - smaller large language models (sLLMs) containing fewer than 10 billion parameters, which can be easily fine-tuned locally, and bigger large language models (bLLMs) with over 10 billion parameters, which we adopt in-context learning. sLLMs are usually pre-trained to learn context-aware word representations. These representations are very effective when used as general-purpose semantic features. sLLMs have largely raised the performance bar of many natural language processing (NLP) tasks, e.g., SA [188], machine translation. They follow the "pre-training and fine-tuning" learning paradigm. To work on downstream tasks, they require fine-tuning on domain-specific data. bLLMs usually contain billions of parameters and exhibit strong capacities to understand natural language and solve complex tasks by text generation. bLLMs adopts "in-context learning" paradigm: they can be adapted to a downstream task simply by providing a prompt (a natural language description of the task) [36].

In summary, this dissertation demonstrates how LLMs can automate different SE tasks involving classification, ranking, and generation tasks. This research aims to improve the accuracy and efficiency of software text analysis, ultimately benefiting the SE community.

1.1 Thesis Statement

The thesis statement of this dissertation is as follows:

> We can leverage large language models to support software developers in automating key tasks such as (1) detecting sentiment polarities, (2) identifying potential duplicate BRs, and (3) crafting pull request titles.

Specifically, one publication aims to identify the sentiments expressed in software-related texts [187]. Similarly, one publication works on detecting duplicate BRs [183], and another focuses on automating pull request title generation [184] to improve software development efficiency. Furthermore, two submissions are currently under review: one revisits the SA task in SE with the recently proposed bLLMs [186], and the other combines the bLLMs with a traditional information-retrieval tool for duplicate BR detection [185].

This dissertation makes the following contribution:

- We leverage various LLMs for the SA4SE task. First, we provide a large-scale comparative analysis between five existing SA4SE tools and four sLLMs on six SE datasets and demonstrate that sLLMs perform better than prior specialized SA4SE tools. Furthermore, we examine the effectiveness of open-source bLLMs on the SA4SE task. We evaluate three open-source bLLMs under zero and few-shot settings. We compare fine-tuned sLLMs with bLLMs on five SE datasets collected from five distinct platforms.

- We conduct a benchmark study about bias in DBRD data and propose a new approach for DBRD. First, we investigate the significance of age bias, state bias, and issue tracking system (ITS) bias in evaluating DBRD techniques. The result depicts that age bias and ITS bias have significant impacts, while state bias has no significant impact. Next, we provide a rich dataset containing recent three-year BRs from Bugzilla, Jira, and GitHub ITSs of six projects. We evaluate state-of-the-art DBRD research tools on a revised dataset that addresses age and ITS biases, and the result shows a simple retrieval-based approach can beat recently proposed sophisticated deep learning-based models. We also compare state-of-the-art DBRD research tools with two industrial tools, i.e., (1) full-text search (`FTS`) implemented in Bugzilla ITS and used by `Mozilla`, (2) `VSCodeBot` used by `VSCode` repository. The result shows that FTS outperforms some research tools on some projects. The best-performing research tools, however, can outperform FTS by 22.1% to 62.7% in terms of Recall Rate@10. `VSCodeBot` is better than most research tools, but the best research tool can outperform it by 9.8% in terms of Recall Rate@5. Furthermore, we propose Cupid, which combines bLLMs with the traditional DBRD technique to enhance the accuracy of DBRD in software systems with the typical number of BRs. We evaluate Cupid on three datasets from open-source projects and compare Cupid with three prior state-of-the-art approaches. The experimental results indicate that Cupid surpasses the performance of these existing DBRD approaches. Notably, Cupid achieves Recall Rate@10 scores ranging from 0.59 to 0.67 across all the datasets analyzed.

- We introduce the task of automatic PR title generation. We construct a dataset named PRTiger, which consists of 43,816 PRs from GitHub for the PR title generation task. PRTiger is the first benchmark for this task. We conduct evaluation of state-of-the-art summarization methods, including extractive (Oracle extraction, BERT-SumExt [106]) and abstractive ones (PRSummarizer [107], iTAPE [46], BART [94],

and T5 [132]). The evaluation includes both automatic evaluation and manual evaluation from multiple aspects. The results show that BART outperforms other approaches by significant margins.

1.2 Sentiment Analysis

SA is a computational study of people's opinions, attitudes, and emotions toward an entity, which can be an individual, an event, or a topic [112]. SA4SE has drawn much attention in recent years [74, 85, 164, 79, 22, 80, 123, 99, 121]. Most research considers SA to be a sentiment polarity classification task. For a given text unit, the goal is to determine its sentiment orientation, i.e., negative, neutral, or positive.

Prior studies have demonstrated that general SA tools work well on social media posts or product reviews while performing poorly on SE datasets [85, 157]. This discrepancy has spurred a growing interest in developing SE-specific SA tools over the past decade [22, 47, 82]. These SA4SE tools usually either propose a SE-specific lexicon [82] or a SE-specific model [22]. At the same time, several benchmarking studies on evaluating general SA tools and SE-specific tools have been conducted [99, 187, 123, 85].

1.3 Duplicate Bug Report Detection

As software systems become larger and more complex, they inevitably contain bugs. *Bug reports* (BRs) are the main channel for users to report bugs to developers. Most software projects use issue tracking systems (ITSs), such as Bugzilla [11], Jira [16] and GitHub [13], to manage BRs and track the progress of bug fixing. Users can submit a BR to the issue tracking system when they find a bug. Then, the developers will fix the bug according

to the description in the BR. However, many BRs are duplicates of the existing BRs. For example, in the dataset constructed by Lazar et al. [93], duplicate BRs represent 12.67% - 23% out of the total BRs in a system.

The existence of duplicate BRs may cost extra software maintenance efforts in bug triage and fixing [91]. In practice, duplicate BRs can also be hard to identify. For example, prior research [134] found duplicate BRs taking thousands of days to identify, with up to 230 comments, involving up to 75 people. This kind of BRs consumed much effort before it was finally identified as duplicate. Identifying duplicate BRs as soon as possible is crucial to avoid wasting developers' time and effort on fixing the same bug multiple times. To alleviate the heavy burden of triagers and decrease the cost of software maintenance, many automatic techniques have been proposed in the past decade [153, 191, 23, 138]. Among them, DBRD techniques have been deployed in practice. For example, some ITSs provide recommendations of potential duplicates to a reporter before submitting a BR. Others employ a bot that flags a duplicate after it has been submitted. For both scenarios, a ranked list of potential duplicate BRs is produced for manual inspection.

1.4 Pull Request Title Generation

As an emerging paradigm, pull-based software development has been widely applied in distributed software development [67, 176]. With the pull-based development model, the project's main repository is not shared for direct modification among *contributors* [69]. Instead, contributors fork or clone the repository and make changes on their own branch. When their changes are ready to be merged into the main branch, they create Pull Requests (PRs); the *core team* (i.e., the *integrators*) then review the changes made in the PRs to make sure that the changes satisfy functional (e.g., no compile error and test failure) and non-functional requirements (e.g., abide to coding convention). They may propose corrections,

discuss with the contributors, and eventually accept or reject the changes [68]. PRs are utilized in nearly half of the collaborative projects in GitHub [69]. Pull-request-based development is usually associated with reduced review times and many contributions [192].

Generally, a PR consists of a title, a description (optional), and a set of commits. The PR title and description are designed to help the *readers* (not limited to integrators, but refers to anyone reading the PR) grasp the context and purpose of the PR. Frequently, a PR is linked with one or more issue reports. Issue reports in issue tracking systems (e.g., GitHub issues) are used to keep track of bugs, enhancements, or other requests. Therefore, many PRs contain the identifiers of the linked issues in their titles or descriptions.

1.5 Dissertation Organization

The remainder of this dissertation is organized as follows.

Chapter 2: Sentiment Analysis

This chapter presents our study on the SA of SE artifacts. We have one published work on leveraging sLLMs for SA; the other work is under review, which compares the sLLMs and bLLMs on this task.

Chapter 3: Duplicate Bug Report Detection

This chapter presents the study we conducted on detecting duplicate BRs. We have one published work on benchmarking DBRD, and the other is under review, introducing a new approach to more accurate DBRD.

Chapter 4: Pull Request Title Generation

This chapter provides the first study on automatically generating pull request titles.

Chapter 5: Related Work

This chapter presents an overview of related work on SA, duplicate bug report detection, and pull request title generation.

Chapter 6: Conclusion and Future Work

This chapter presents an overview of related work on the four tasks investigated.

Chapter 2

Sentiment Analysis

This chapter presents two empirical studies on SA4SE, i.e, in Section 2.1, we compare sLLMs with prior SA4SE tools, and in Section 2.2, we compare sLLMs and bLLMs.

2.1 How Prior SA4SE Tools Compare to sLLMs?

To understand the performance of SA4SE tools, three benchmarking studies have been conducted: Lin et al. [99] compared five SA tools, i.e., SentiStrength, NLTK, Stanford CoreNLP, SentiStrength-SE, and Stanford CoreNLP SO on three datasets – mobile APP reviews, Stack Overflow posts, and Jira issue comments. Regarding the number of correct predictions, SentiStrength-SE, Stanford CoreNLP, and SentiStrength perform the best for one of the datasets. Islam et al. [80] compared three SA4SE tools, i.e., SentiStrength-SE, Senti4SD, and EmoTxt, on three datasets – Jira issue comments, Stack Overflow posts, and code review comments. SentiStrength-SE achieved the highest macro-averaged F1-score for Jira issue comment and code review comment datasets. At the same time, Senti4SD performed the best for the Stack Overflow post dataset. Novielli et al. [123] compared four

tools, i.e., Senti4SD, SentiStrength-SE, SentiCR, and SentiStrength, on four datasets – Stack Overflow posts, Jira issue comments, code reviews comments, and Stack Overflow posts related to Java libraries.[1] They found that Senti4SD achieved the highest macro-averaged F1 score for the Stack Overflow dataset, while SentiCR was the highest for the other three datasets.

Inspired by these three studies, we raise the main research question: *How well can pre-trained sLLMs perform for SA4SE task?* To answer the question, we conduct a large-scale exploratory study. Specifically, we (1) consider a diverse collection of six datasets (instead of three or four considered in prior work), (2) compare the effectiveness of the best performers in prior work [99, 80, 123] with state-of-the-art sLLMs. This study investigates the following specific research questions:

- **RQ1**: *How accurate are sLLMs as compared to existing SA4SE tools?*
- **RQ2**: *How efficient are sLLMs as compared to existing SA4SE tools?*

To answer the above questions, we compare the accuracy and efficiency of the best-performing SA4SE approaches in prior studies against four sLLMs. Prior studies [99, 80, 123] have highlighted that Stanford CoreNLP, SentiStrength, SentiStrength-SE, SentiCR, and SentiSD are the best performers on at least one dataset. For sLLMs, we consider BERT [55], RoBERTa [107], XLNet [175], and ALBERT [92]. We fine-tune these models with labeled SE-specific data for SA4SE tasks.[2] We evaluate the approaches on six datasets: API reviews (`API`), Stack Overflow posts (`StackOverflow`), Mobile APP reviews (`GooglePlay`), GitHub pull-request and commit comments (`GitHub`), Jira issue comments (`Jira`), and Gerrit code review comments (`Gerrit`).

[1] They referred to the dataset as 'Java Libraries' and it is the Stack Overflow dataset from Lin et al.'s work [99].

[2] For brevity, unless otherwise stated, we refer to these fine-tuned sLLMs as sLLMs.

The experimental results demonstrate that in all the six datasets, sLLMs, i.e., BERT, RoBERTa, XLNet, and ALBERT, can achieve better performance than the best performing SA4SE tools identified in prior studies [99, 80, 123]. Across these datasets, sLLMs consistently outperform previous SA4SE tools by 6.5% to 35.6% in terms of macro/micro-averaged F1-scores. This accuracy boost comes with some runtime costs: sLLMs are less efficient than existing SA4SE approaches (except Senti4SD and Stanford CoreNLP). Still, its runtime cost is not prohibitively high; it requires 15 seconds to 10 minutes to fine-tune, while it can predict sentiments of hundreds of text units (documents) in seconds.

2.1.1 Experimented Techniques

Prior SA4SE Tools

In this section, we briefly describe details about the best-performing approaches identified by the prior benchmarking works [99, 80, 123]: Stanford CoreNLP, SentiStrength, SentiStrength-SE, SentiCR, and Senti4SD. We refer to them collectively as the PRIOR group.

Stanford CoreNLP, proposed by Socher et al. [149], is designed for single-sentence sentiment classification; it can return a sentiment value and polarity for a sentence. Socher et al. introduced the Stanford Sentiment Treebank, which includes fine-grained sentiment labels for 215,154 phrases in the parse trees. The parse trees consist of 11,855 sentences extracted from the movie review dataset, initially constructed by Pang and Lee [30]. Socher et al. also proposed a new model called Recursive Neural Tensor Network to capture the compositional effects with higher accuracy. Stanford CoreNLP is trained with this Recursive Neural Tensor Network on the Stanford Sentiment Treebank.

SentiStrength is a lexicon-based approach developed by Thelwall et al. [156]. As a lexicon-

based approach, SentiStrength has several dictionaries, including both formal terms and informal texts (such as emoticons, idioms, and slang). In these dictionaries, each term is labeled with a sentiment strength. Based on these dictionaries and linguistic analysis, given a sentence, SentiStrength will output two integers: one is for *positive* emotion, and the other is for *negative* emotion. It not only categorizes the emotional polarity but also gives the strength of the polarity. The scale for positive emotion is from 1 to 5, representing not positive to very strong positive; the range for negative emotion is from -1 to -5, representing not negative to very strong negative.

SentiStrength-SE, proposed by Islam and Zibran, is a customized version of SentiStrength, implemented by adding a domain-specific dictionary [79]. SentiStrength-SE is also the first SA tool considering SE-specific context, and it is designed based on in-depth qualitative research. Specifically, Islam and Zibran first used SentiStrength to detect sentiment in `Jira` issue comments. They analyzed 151 `Jira` issue comments for which SentiStrength produced wrong outputs. This analysis was performed to identify the reasons/difficulties that affect the accuracy of SentiStrength. Finally, they identified 12 such difficulties. They also found that out of all the difficulties, the domain-specific meanings of words were the most prevalent. To build a domain dictionary, they first collected a large dataset of commit messages drawn from 50 open-source projects from `GitHub` provided by their earlier work [78]. Then, they extracted the lemmatizations of all words in the dataset. Next, they kept the overlap between these word lemmatizations and the SentiStrength dictionary of sentiment words. A total of 716 words remain. Through manual assessments, they further eliminated words that carry neutral sentiments. Finally, the final word dictionary of SentiStrength-SE consists of 500 words, of which 167 are positive and 293 are negative. They also extended the dictionary by adding new sentiment words and negations. Additionally, contextual information is considered in SentiStrength-SE.

SentiCR [22] is designed by Ahmed et al., particularly for code review comments. Based

on the characteristics of code review comments, SentiCR has a suite of data preprocessing steps, including URL removal and code snippet removal. SentiCR includes a two-stage negation preprocessing approach. Ahmed et al. first build a chunk grammar (i.e., a set of rules indicating how sentences should be chunked) for *NLTK RegexpChunkParser* to identify negation phrases. Second, they modify all the verbs, adjectives, and adverbs in a negation phrase identified by the chunker by prepending *not* to it [22]. After generating feature vectors using TF-IDF, eight supervised classifiers are evaluated. They also use 10-fold cross-validation to validate each algorithm. GBT (Gradient Boosting Tree) [128] demonstrates the highest precision, recall, and accuracy among all the eight used algorithms. Thus, by default, the supervised classifier in SentiCR is GBT. The original SentiCR is trained to classify a code review comment as negative or non-negative.

Senti4SD is another supervised learning-based tool. The largest difference between Senti4SD and previous SE-specific tools is how it generates feature vectors. Senti4SD [42] utilizes three features based on (1) Generic sentiment lexicons. It uses SentiStrength lexicons; (2) Keywords (n-gram extracted from the dataset). It primarily uses uni-gram and bi-gram. The value of each keyword feature corresponds to its number of occurrences. In addition to uni-gram and bi-gram, it includes other keyword features, e.g., total occurrences of uppercase words and slang expressions for laughter; (3) Word representation in a distributional semantic model (DSM) specifically trained on Stack Overflow data. DSM uses the CBOW architecture implemented by word2vec [114]. Each Stack Overflow document (i.e., answers, questions, and comments) is represented as the vector sum of all the vectors of words in the document. Besides, it calculates four prototype vectors, namely, *p_pos*, *p_neg*, *p_neu*, and *p_subj*, respectively. *p_pos* is the sum of all the word vectors in each document, which have positive polarity in the SentiStrength lexicon dictionary. Similarly, by summing up all the negative/neutral word vectors in the document, we have *p_neg* and *p_neu*. *p_subj* is the sum of *p_pos* and *p_neg*. Using these four objective vectors

for a document, Senti4SD calculates the similarity scores between document vectors to get the semantic features. Finally, based on the features mentioned above, Senti4SD is trained to distinguish sentiment polarities of text units by using Support Vector Machines (SVM).

sLLMs

We provide a brief introduction to BERT, RoBERTa, XLNet, and ALBERT. We refer to these models collectively as the sLLMs (smaller-size large language models) group. We also present the exact model we used on the Hugging Face platform [168] in the parenthesis. **BERT** (`bert-base-uncased`) [55], which stands for Bidirectional Encoder Representations from Transformers, introduces two key pre-training tasks. The first is mask language modeling (MLM), where BERT learns to predict masked words in a given text. Additionally, BERT incorporates the next sentence prediction (NSP) task, training to determine whether the second sentence logically follows the first or is a random sentence from the training data.

RoBERTa (`roberta-base`) [107] is short for "A Robustly Optimized BERT Pretraining Approach". RoBERTa is a BERT variant distinguished by its innovative training strategies and hyperparameter choices. Notably, it eliminates the NSP task, employs a larger batch size, trains on a larger corpus than BERT, and utilizes a dynamic masking strategy during training.

ALBERT (`albert-base-v2`) [92], or "A Lite BERT", is another BERT variant designed to reduce model size and computational requirements while maintaining or improving performance. ALBERT retains the MLM task but replaces the NSP task with the sentence order prediction (SOP) task. In SOP, ALBERT is trained to predict whether pairs of sentences are correctly ordered or if their positions have been swapped.

Table 2.1: Investigated datasets

dataset	# doc	# (%) positive	# (%) neutral	# (%) negative
API	4,522	890 (19.7)	3,136 (69.3)	496 (11)
StackOverflow	1,500	131 (8.7)	1,191 (79.4)	178 (11.9)
GooglePlay	341	186 (54.5)	25 (7.3)	130 (38.1)
GitHub	7,122	2,013 (28.3)	3,022 (42.4)	2,087 (29.3)
Jira	926	290 (31.3)	-	636 (68.7)
	# doc	**# (%) non-negative**		**# (%) negative**
Gerrit	1,600	1,202 (75.1)		398 (24.9)

XLNet (`xlnet-base-cased`) [175] primarily focuses on capturing contextual information and long-range dependencies in text. It employs an autoregressive pretraining method and introduces permutation language modeling, where word order in a sentence is randomly shuffled, and the model is trained to predict the original sequence. XLNet also incorporates innovations such as the "two-stream self-attention" mechanism.

2.1.2 Study Setup

This section first describes the six datasets used in this work, and defines the SA task based on the polarity labels in the datasets. Then, we elaborate on the implementations of all the considered approaches. Lastly, we describe the relevant evaluation metrics and settings.

Datasets

In this comparative study, we use six publicly available datasets with annotated sentiment polarities. Table 2.1 shows the detailed statistics of the six datasets, including the total number of documents in a dataset (# doc) and the number (and percentage) of documents with one of the sentiment polarities (e.g., # (%) positive, # (%) neutral, # (%) negative).

API reviews (API) [161]. It includes 4,522 sentences from 1,338 Stack Overflow posts. This dataset contains both API aspects and the polarities of provided opinions, i.e., positive, negative, and neutral), curated by Uddin and Khomh.

Besides this API dataset, we experimented with the existing manually labeled datasets from five distinct platforms: Gerrit, GitHub, Google Play, Jira, and Stack Overflow. For simplicity, we refer to these datasets using abbreviations: Gerrit, GitHub, GooglePlay, Jira, and StackOverflow.

Gerrit Dataset: Ahmed et al. [22] meticulously labeled this dataset. They initiated their process by mining code review repositories from 20 prominent open-source software (OSS) projects. Three raters individually labeled the selected code review comments and resolved conflicts through discussion. The dataset was refined into two classes: *negative* and *non-negative*, forming the final dataset.

GitHub Dataset: Novielli et al. [121] curated the GitHub dataset, which comprises pull request and commit comments. Sentiment was assessed based on the entire comment, rather than isolated portions. The labeling process began with the manual classification of 4,000 comments, followed by a semi-automatic approach using Senti4SD [42], which required manual confirmation of the automatically assigned polarity labels.

GooglePlay, Jira, and StackOverflow Datasets: Lin et al. [99] provided three datasets, each with its unique characteristics:

- **GooglePlay Dataset**: Originally collected by Chen et al. [45], this dataset contains user reviews of Android apps on Google Play. Villarroel et al. [164] selected a subset of reviews from Chen et al.'s dataset, and Lin et al. further sampled 341 reviews. Lin et al. performed the manual labeling of sentiment, where two annotators indi-

vidually classified text as positive, neutral, or negative. In cases of disagreement, a third evaluator was involved for resolution.

- **`Jira` Dataset**: This dataset comprises `Jira` issue comment sentences and was originally collected and labeled by Ortu et al. [126]. However, Ortu et al.'s dataset only provided emotional labels, such as *love*, *joy*, *anger*, and *sadness*. Lin et al. mapped sentences labeled with *love* or *joy* to "positive" and those labeled with *anger* or *sadness* to "negative".

- **`StackOverflow` Dataset**: Lin et al. gathered and labeled the `StackOverflow` dataset, extracting 5,073,452 sentences from the latest Stack Overflow dump available in July 2017. The sentences were selected based on two criteria: they had to be tagged with "Java", and they needed to contain keywords such as "library/libraries" or "`API` (s)". A random sample of 1,500 sentences was manually labeled by assigning a sentiment score to each sentence. Two annotators did the manual annotation individually, with conflicts resolved through discussion.

We considered all the three datasets used in Lin et al.'s benchmarking work [99] – mobile APP reviews (`GooglePlay`), Stack Overflow posts (`StackOverflow`), and `Jira` issue comments (`Jira`). We include another three datasets (i.e., `GitHub`, `API`, and `Gerrit`) with diverse characteristics in at least four aspects. First, the added three datasets were constructed from various repositories: pull-request and issue comments from GitHub [121]; API reviews from Stack Overflow [161]; and Gerrit code review comments from open-source projects [22]. Second, the sizes of the three datasets differ significantly. For example, the `GitHub` dataset is over 20 times larger than the mobile APP reviews. Third, among the three datasets, the `GitHub` dataset is balanced in the number of positive, neutral, and negative text units, while the other two are imbalanced. Fourth, unlike the other two, the code review comment dataset only has two sentiment polarities: non-negative and

negative.

For a given text unit, each approach will predict its sentiment polarity label. According to the number of sentiment polarities in a dataset, we formulate the problem as a binary or ternary text classification task. Specifically, the classification tasks for the datasets `Jira` and `Gerrit` correspond to binary classification tasks. Both datasets have two polarity labels: positive and negative for `Jira`; negative and non-negative for `Gerrit`. For the other four datasets, the classification problems are formulated as ternary-class classification tasks as they have three polarity labels, i.e., positive, neutral, and negative.

Implementations

SA4SE Tools: For *Stanford CoreNLP*[3], we used its Python wrapper[4]. Given a sentence, Stanford CoreNLP returns the sentiment polarity with its corresponding sentiment value (*Very negative*=0, *Negative*=1, *Neutral*=2, *Positive*=3, *Very positive*=4). As Stanford CoreNLP gives a sentiment value and polarity to individual sentences, when a text unit in some datasets has more than one sentence, we calculate the average sentiment value of all sentences for the text unit. If the average sentiment value of a text unit is greater than 2, we assign it a *positive* polarity; if the value is less than 2, we assign it a *negative* polarity; otherwise, we assign it a *neutral* polarity (c.f. [99]).

For *SentiStrength*[5] and *SentiStrength-SE*[6], following Lin et al., we sum up the two sentiment strength scores returned by the tool to get the overall polarity for a sentence. If the total score is greater than 0, we assign a *positive* polarity to the whole text unit; if the total score

[3] http://stanfordnlp.github.io/CoreNLP/

[4] https://github.com/smilli/py-corenlp/

[5] http://sentistrength.wlv.ac.uk/download.html

[6] https://laser.cs.uno.edu/Projects/Projects.html

is less than 0, we assign a *negative* polarity, and a *neutral* polarity is for the text unit that has a total score of 0. For the code review comment dataset, we need to distinguish between *non-negative* and *negative*; hence, if the overall score is less than 0, we assign *negative* to the text unit; otherwise, we assign *non-negative* to it.

SentiCR[7] only classifies two polarities, i.e., negative and non-negative. We re-train it on each dataset to classify three polarities, i.e., positive, neutral, and negative. We only changed the training dataset and kept all the parameters as default.

For *Senti4SD*, we use the classifier pre-trained on Stack Overflow dataset[8]. Senti4SD can classify three polarities, i.e., positive, neutral, and negative. As the code review comment dataset only has *negative* and *non-negative* polarities, we assign both the *positive* and *neutral* as *non-negative*.

sLLMs. Many existing sLLMs are pre-trained for general domains. For instance, a combination of BooksCorpus [193] and English Wikipedia is used as all or part of the BERT and XLNet pre-training corpus. To build a sentiment classification model, we add a feed-forward dense layer and softmax activation function on top of each model. A certain pre-trained model's parameters have been reused as a starting point. We feed our SE training data to a pre-trained sLLM's tokenizer and get the required formatted data; Then, we use the formatted data to train the pre-trained model further to get a fine-tuned model. Finally, we test it on the held-out test data. As found in BERT paper [55], the following values of hyper-parameter for BERT fine-tuning procedure work well across all tasks: (1) Batch size: 16, 32; (2) Number of epochs: 2, 3, 4; (3) Learning rate (Adam): 5e-5, 3e-5, 2e-5. For all these models, we run them in 4 epochs with a batch size of 16. Moreover, we set the learning rate to 2e-5. We used AdamW optimizer.

[7] https://github.com/senticr/SentiCR

[8] https://github.com/collab-uniba/Senti4SD

Evaluation Metrics

Following the previous work [123], we report the precision, recall, and F1-score of each approach for each polarity. We also report the macro- and micro-averaged metrics to show overall multi-classification performances. The formula to calculate P (precision), R (recall) and $F1$ (F1-score) are as follows: $P = \frac{TP}{TP+FP}$,$R = \frac{TP}{TP+FN}$, and $F1 = 2 \cdot \frac{P \cdot R}{P+R}$. TP refers to the number of true positives (text units correctly classified as positive), FP refers to the number of false positives (text units mistakenly classified as positive), and FN refers to false negatives (text units mistakenly classified as negative).

The macro-averaged metric regards the measurement of each sentiment class equally. It takes the precision, recall, and F1-score of each class and then averages them. The micro-averaged metric calculates measurement over all data points in all classes and tends to be mainly influenced by the performance of the majority class [123]. The formulas for macro- and micro-averaged precision (P) are shown below:

$$P_{macro} = \frac{\sum_{i=1}^{k} P_i}{k} \tag{2.1}$$

$$P_{micro} = \frac{\sum_{i=1}^{k} TP_i}{\sum_{i=1}^{k} TP_i + \sum_{i=1}^{k} FP_i} \tag{2.2}$$

P_{macro} and P_{micro} represent macro- and micro-averaged precision respectively. P_i, TP_i and FP_i represent the precision, number of true positives, and number of false positives for the ith class, respectively. k denotes the number of sentiment polarity classes. We can calculate macro- and micro-averaged recall and F1, denoted as R_{macro}, R_{micro}, $F1_{macro}$, $F1_{micro}$, similarly. We consider a model is better than another only when it achieves higher values of both $F1_{macro}$ and $F1_{micro}$.

Experimental Setting

Following Novielli et al. [123], we split each dataset into a training set (70%) and a test set (30%). Since SentiCR is originally designed for binary classification, we re-train it using the training set and test it on the test set for three classes. For Senti4SD, SentiStrength, and SentiStrength-SE, they do not need re-training. Concerning the four sLLMs, we fine-tune them with the training set and then evaluate them on the test set.

2.1.3 Study Results

In this section, we report the performance of the nine SA approaches on the six datasets described in Section 2.1.2. For each dataset, we highlight the best performance of the two main metrics (i.e., macro- and micro-averaged F1-scores) in **bold**. We answer the research questions based on the experimental results as follows.

RQ1: How accurate are sLLMs as compared to existing SA4SE tools?

To answer RQ1, we compare all the nine approaches in both the PRIOR and sLLM groups. Tables 2.2 and 2.3 present the performance of the nine approaches on the six datasets.

`API` and `StackOverflow` Datasets: Similar to the `StackOverflow` dataset, the `API` dataset is constructed from Stack Overflow posts. Thus, we look at the results of both datasets together. In terms of both macro- and micro-averaged F1, the approaches in sLLM group outperform those in the PRIOR group. For the `API` dataset, The best performing sLLM approach (ALBERT) can achieve macro- and micro-averaged F1-scores of 0.82 and 0.89, respectively. On the other hand, the best-performing PRIOR approach (SentiCR) can only achieve macro- and micro-averaged F1-scores of 0.66 and 0.82, respectively. We observe a

Table 2.2: Results for `API`, `StackOverflow`, `GooglePlay`, and `GitHub` Datasets

Dataset	Approach	Positive			Neutral			Negative			Macro-avg			Micro-avg		
		P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
`API`	**Stanford CoreNLP**	0.47	0.41	0.44	0.85	0.60	0.71	0.22	0.66	0.33	0.51	0.56	0.49	0.57	0.57	0.57
	SentiStrength	0.44	0.45	0.45	0.81	0.77	0.79	0.44	0.45	0.45	0.55	0.57	0.56	0.68	0.68	0.68
	SentiStrength-SE	0.59	0.33	0.42	0.77	0.91	0.83	0.47	0.26	0.33	0.61	0.50	0.53	0.73	0.73	0.73
	SentiCR	0.85	0.52	0.65	0.82	0.98	0.89	0.81	0.31	0.45	0.83	0.61	**0.66**	0.82	0.82	**0.82**
	Senti4SD	0.56	0.33	0.41	0.76	0.93	0.84	0.44	0.10	0.17	0.59	0.45	0.47	0.73	0.73	0.73
	BERT	0.85	0.72	0.78	0.92	0.95	0.93	0.73	0.73	0.73	0.83	0.80	0.81	0.89	0.89	**0.89**
	RoBERTa	0.78	0.79	0.78	0.93	0.93	0.93	0.72	0.70	0.71	0.81	0.81	0.81	0.88	0.88	0.88
	XLNet	0.75	0.75	0.75	0.91	0.91	0.91	0.63	0.59	0.61	0.76	0.75	0.76	0.85	0.85	0.85
	ALBERT	0.88	0.77	0.82	0.92	0.96	0.94	0.71	0.68	0.70	0.84	0.80	**0.82**	0.89	0.89	**0.89**
`StackOverflow`	**Stanford CoreNLP**	0.23	0.42	0.30	0.92	0.69	0.79	0.34	0.82	0.48	0.50	0.64	0.52	0.68	0.68	0.68
	SentiStrength	0.25	0.42	0.32	0.89	0.80	0.84	0.40	0.52	0.46	0.52	0.58	0.54	0.74	0.74	0.74
	SentiStrength-SE	0.31	0.13	0.19	0.83	0.94	0.89	0.44	0.18	0.26	0.53	0.42	0.44	0.80	0.80	0.80
	SentiCR	0.48	0.32	0.38	0.90	0.90	0.90	0.45	0.55	0.49	0.61	0.59	**0.59**	0.82	0.82	0.82
	Senti4SD	0.50	0.34	0.41	0.85	0.96	0.90	0.75	0.14	0.23	0.70	0.48	0.51	0.83	0.83	**0.83**
	BERT	0.65	0.63	0.64	0.94	0.95	0.94	0.73	0.68	0.71	0.77	0.75	0.76	0.90	0.90	**0.90**
	RoBERTa	0.57	0.76	0.65	0.96	0.92	0.94	0.78	0.82	0.80	0.77	0.83	**0.80**	0.90	0.90	**0.90**
	XLNet	0.50	0.76	0.60	0.96	0.90	0.93	0.74	0.84	0.79	0.73	0.83	0.77	0.88	0.88	0.88
	ALBERT	0.71	0.32	0.44	0.90	0.95	0.92	0.61	0.61	0.61	0.74	0.63	0.66	0.86	0.86	0.86
`GooglePlay`	**Stanford CoreNLP**	0.77	0.68	0.72	0.14	0.43	0.21	0.69	0.54	0.61	0.53	0.55	0.51	0.61	0.61	0.61
	SentiStrength	0.75	0.90	0.82	0.12	0.29	0.17	0.73	0.30	0.42	0.53	0.49	0.47	0.64	0.64	0.64
	SentiStrength-SE	0.73	0.81	0.77	0.15	0.57	0.24	0.91	0.27	0.42	0.60	0.55	0.48	0.60	0.60	0.60
	SentiCR	0.86	0.83	0.84	0.00	0.00	0.00	0.68	0.81	0.74	0.51	0.55	**0.53**	0.77	0.77	**0.77**
	Senti4SD	0.72	0.85	0.78	0.12	0.29	0.17	0.65	0.30	0.41	0.50	0.48	0.45	0.61	0.61	0.61
	BERT	0.86	0.95	0.90	0.00	0.00	0.00	0.87	0.89	0.88	0.58	0.61	0.59	0.86	0.86	0.86
	RoBERTa	0.95	0.92	0.93	0.00	0.00	0.00	0.84	1.00	0.91	0.60	0.64	**0.61**	0.88	0.88	**0.88**
	XLNet	0.87	0.98	0.92	0.00	0.00	0.00	0.86	0.81	0.83	0.57	0.60	0.58	0.85	0.85	0.85
	ALBERT	0.91	0.86	0.89	0.00	0.00	0.00	0.72	0.92	0.81	0.54	0.59	0.57	0.83	0.83	0.83
`GitHub`	**Stanford CoreNLP**	0.61	0.36	0.45	0.44	0.40	0.42	0.40	0.61	0.48	0.48	0.46	0.45	0.45	0.45	0.45
	SentiStrength	0.65	0.66	0.66	0.60	0.58	0.59	0.63	0.66	0.65	0.63	0.63	0.63	0.63	0.63	0.63
	SentiStrength-SE	0.87	0.85	0.86	0.77	0.86	0.81	0.82	0.71	0.76	0.82	0.81	0.81	0.81	0.81	0.81
	SentiCR	0.88	0.86	0.87	0.78	0.91	0.84	0.86	0.68	0.76	0.84	0.82	**0.82**	0.83	0.83	**0.83**
	Senti4SD	0.79	0.84	0.82	0.69	0.86	0.76	0.82	0.47	0.60	0.77	0.73	0.73	0.74	0.74	0.74
	BERT	0.92	0.95	0.93	0.90	0.92	0.91	0.93	0.87	0.90	0.92	0.91	**0.92**	0.92	0.92	**0.92**
	RoBERTa	0.93	0.96	0.94	0.91	0.92	0.92	0.93	0.89	0.91	0.93	0.92	**0.92**	0.92	0.92	**0.92**
	XLNet	0.90	0.97	0.94	0.94	0.89	0.91	0.91	0.92	0.91	0.92	0.93	**0.92**	0.92	0.92	**0.92**
	ALBERT	0.91	0.93	0.92	0.85	0.94	0.89	0.94	0.78	0.85	0.90	0.88	0.89	0.89	0.89	0.89

similar finding for the `StackOverflow` dataset.

`GooglePlay` Dataset: We find that all the approaches perform relatively poorly on the `GooglePlay` dataset. One potential reason is that this dataset is highly imbalanced and quite small; there are only a few text units with a neutral sentiment. Due to this limited number of text units for training, the approaches that have been trained on this dataset (i.e., all sLLM approaches and SentiCR) have worse performance than the lexicon-based

Table 2.3: Results for `Jira` and `Gerrit` Datasets

Dataset	Approach	Positive			Negative			Macro-avg			Micro-avg		
		P	R	F1	P	R	F1	P	R	F1	P	R	F1
`Jira`	**Stanford CoreNLP**	0.83	0.50	0.62	0.96	0.63	0.76	0.60	0.38	0.46	0.58	0.58	0.58
	SentiStrength	0.95	0.91	0.93	0.99	0.72	0.83	0.65	0.54	0.59	0.78	0.78	0.78
	SentiStrength-SE	0.98	0.85	0.91	0.99	0.54	0.70	0.66	0.46	0.54	0.65	0.65	0.65
	SentiCR	0.96	0.81	0.88	0.90	0.98	0.94	0.93	0.89	**0.91**	0.92	0.92	**0.92**
	Senti4SD	0.90	0.86	0.88	1.00	0.21	0.34	0.63	0.35	0.41	0.44	0.44	0.44
	BERT	0.99	0.96	0.97	0.98	0.99	0.99	0.98	0.98	**0.98**	0.98	0.98	**0.98**
	RoBERTa	0.98	0.96	0.97	0.98	0.99	0.98	0.98	0.97	**0.98**	0.97	0.97	0.97
	XLNet	0.98	0.96	0.97	0.98	0.99	0.98	0.98	0.97	**0.98**	0.98	0.98	**0.98**
	ALBERT	0.97	0.94	0.95	0.97	0.98	0.98	0.97	0.96	0.96	0.97	0.97	0.97
Dataset	**Approach**	**Non-negative**			**Negative**			**Macro-avg**			**Micro-avg**		
		P	R	F1	P	R	F1	P	R	F1	P	R	F1
`Gerrit`	**Stanford CoreNLP**	0.91	0.55	0.69	0.37	0.83	0.51	0.64	0.69	0.60	0.62	0.62	0.62
	SentiStrength	0.81	0.82	0.82	0.41	0.40	0.41	0.61	0.61	0.61	0.72	0.72	0.72
	SentiStrength-SE	0.80	0.94	0.86	0.57	0.25	0.35	0.68	0.59	0.60	0.77	0.77	0.77
	SentiCR	0.87	0.83	0.85	0.54	0.62	0.58	0.71	0.73	**0.72**	0.78	0.78	**0.78**
	Senti4SD	0.78	0.97	0.86	0.60	0.16	0.25	0.69	0.56	0.56	0.77	0.77	0.77
	BERT	0.94	0.87	0.90	0.67	0.83	0.74	0.80	0.85	0.82	0.86	0.86	0.86
	RoBERTa	0.92	0.93	0.92	0.76	0.74	0.75	0.84	0.83	**0.84**	0.88	0.88	**0.88**
	XLNet	0.87	0.95	0.91	0.78	0.54	0.64	0.82	0.75	0.77	0.85	0.85	0.85
	ALBERT	0.90	0.84	0.87	0.59	0.72	0.65	0.75	0.78	0.76	0.81	0.81	0.81

approaches (i.e., SentiStrength and SentiStrength-SE) for the neutral sentiment. Still, overall, we observe that approaches in the sLLM group outperform those in the PRIOR group.

`GitHub` Dataset: Among all the six datasets, `GitHub` is the largest and most balanced one. The four approaches from the sLLM group achieved similar performance: BERT, RoBERTa, and XLNet produce the same macro- and micro-averaged F1-scores; ALBERT performs slightly worse, 0.03 lower than the other three approaches. Their performance is better than that for approaches in the PRIOR group. In the PRIOR group, SentiCR is the best performer, with SentiStrength-SE being a close second. The other three approaches produce substantially lower macro- and micro-averaged F1-scores. Stanford CoreNLP has

Table 2.4: Comparison between the Best Performers in the PRIOR and sLLM Groups

Metric	Group	API	StackOverflow	GooglePlay	GitHub	Jira	Gerrit
Macro-avg F1	Best **PRIOR**	0.66	0.59	0.53	0.82	0.91	0.72
	Best **sLLM**	0.82	0.80	0.61	0.92	0.98	0.84
	Improvement	24.2%	35.6%	15.1%	12.2%	7.7%	16.7%
Micro-avg F1	Best **PRIOR**	0.82	0.83	0.77	0.83	0.92	0.78
	Best **sLLM**	0.89	0.90	0.88	0.92	0.98	0.88
	Improvement	8.5%	8.4%	14.3%	10.8%	6.5%	12.8%

the worst performance, which shows that Stanford CoreNLP has poor generalization of SE data across different repositories.

Jira Dataset: For the Jira dataset, we found that all the sLLM approaches perform well with high macro- and micro-averaged F1-scores (≥ 0.96). However, in the PRIOR group, only SentiCR achieves macro- and micro-averaged F1-scores greater than 0.90. The other approaches in the PRIOR group have macro-averaged F1-scores lower than 0.6, and micro-averaged F1-scores lower than 0.79. Two noteworthy facts are that SentiStrength outperforms SentiStrength-SE by 20% in terms of the micro-averaged F1-score. Also, Stanford CoreNLP outperforms Senti4SD by 31.8%. This shows that SE-specific SA tools do *not* always outperform general-purpose ones in SE datasets. The performance of different approaches from the PRIOR group gives us another insight: SentiStrength and SentiStrength-SE are both lexicon-based and do not need training. They outperform Stanford CoreNLP and Senti4SD, which have been trained in other datasets. SentiCR, which has been re-trained in this Jira dataset, achieves the best result in the PRIOR group. This highlights that the lexicon-based approaches may be better than supervised ones (if training is not done on a suitable dataset).

Gerrit Dataset: For the dataset Gerrit, all the approaches perform better in detecting non-negative polarity than negative polarity, with each sLLM approach outperforming all PRIOR approaches. Also, all the approaches trained on the Gerrit dataset, including all

sLLM approaches and SentiCR outperform the other four non-`Gerrit` specific tools (i.e., their training and construction did not involve `Gerrit`-datasets).

Overall: We found that the best and worst-performing approaches differ for different datasets. Also, no approach can achieve the best performance on all datasets. For example, RoBERTa achieves the highest micro-averaged F1-score while SentiStrength-SE has the lowest score on the `GooglePlay` dataset. On `API` dataset, BERT and ALBERT achieve the highest micro-averaged F1-score, while Stanford CoreNLP has the lowest score. Also, the performance gap between the best and worst performance on different datasets varies. The difference between micro-averaged F1-scores ranges from 32.4% (on `StackOverflow`) to 122.7% (on `Jira`). For macro-averaged F1-scores, the difference is from 35.6% (on `GooglePlay`) to 139% (on `Jira`).

Among all the approaches in the PRIOR group, we found that SentiCR achieves the best performance on five out of six datasets except `StackOverflow`. Also, Stanford CoreNLP performs the worst on five out of six datasets except on `Jira`. Among the approaches in the sLLM group, we found that RoBERTa performs best on four datasets, i.e., `GooglePlay`, `GitHub`, `StackOverflow`, and `Gerrit`. ALBERT performs the worst on `GooglePlay`, `GitHub`, `StackOverflow`, and `Gerrit`, but it performs best on `API`.

We also observed that all the sLLM approaches outperform PRIOR approaches up to 35.6% in terms of macro- and micro-averaged F1-scores (see Table 2.4). This demonstrates the effectiveness of the sLLM approaches.

RQ1 Main Findings: The sLLMs outperform the prior SA4SE tools consistently across the six datasets, although the best-performing model differs across different datasets. The improvements achieved by the sLLMs range from 6.5% to 35.6% in terms of macro- and micro-averaged F1-scores.

Table 2.5: Training (or fine-tuning) and prediction time (seconds)

Approach	API		StackOverflow		GooglePlay		GitHub		Jira		Gerrit	
	Train	Pred	Train	Pred	Train	Pred	Train	Pred	Train	Pred	Train	Pred
BERT	212.61	8.17	68.18	2.69	15.80	0.63	328.79	13.01	43.39	1.64	73.65	3.06
RoBERTa	215.02	7.87	71.77	2.62	15.91	0.59	338.64	12.42	43.78	1.61	76.35	2.91
XLNet	375.28	18.40	126.15	6.14	28.01	1.39	590.44	29.04	26.82	3.77	154.89	9.22
ALBERT	199.76	7.95	66.74	2.64	14.91	0.61	315.45	12.55	40.81	1.62	72.10	2.91
SentiCR	74.85	1.79	5.31	0.51	0.96	0.17	137.32	3.44	0.72	0.33	3.80	0.86
Senti4SD	-	48.81	-	31.11	-	23.59	-	63.95	-	27.88	-	31.62
Stanford CoreNLP	-	283.39	-	28.63	-	11.29	-	418.65	-	11.89	-	280.59
SentiStrength	-	<1	-	<1	-	<1	-	<1	-	<1	-	<1
SentiStrength-SE	-	1.69	-	<1	-	<1	-	3.22	-	<1	-	<1

RQ2: ***How efficient are sLLMs as compared to existing SA4SE tools?***

The time efficiency of SA4SE approaches can be a concern in practice. Thus, we report the training (fine-tuning for sLLM approaches) and prediction time of all the approaches. Prediction time covers the time from processing the data to output the predicted label. Here, we provide a manual estimation of the exact prediction time of SentiStrength and SentiStrength-SE as they use a graphical user interface.

We run all the approaches on a desktop computer with Nvidia GeForce RTX 2080 Ti and Intel(R) Core(TM) i7-9700K CPU @ 3.60GHz. The sLLM group runs with both GPU and CPU, which the PRIOR group only uses CPU. All approaches, except for SentiStrength, are running with Ubuntu 18.04.4 LTS. SentiStrength runs on a Windows 10 virtual machine on the Ubuntu system, because only its .exe is available online.

In Table 2.5, we report: the training (fine-tuning) time (in seconds) for each training set; and the prediction time (in seconds) on each test set. For approaches in the sLLM group, in terms of both fine-tuning and prediction time, XLNet takes the most time (approximately double the time used by the other three approaches). For the approaches in the PRIOR group, in terms of prediction time, Stanford CoreNLP is the most expensive and SentiCR runs the fastest. Generally, the prediction time used by sLLMs is two times more than that

of SentiStrength, SentiStrength-SE, and SentiCR. However, it is less than 50% of the time used by Senti4SD [9] and Stanford CoreNLP.

RQ2 Main Findings: In general, training (fine tuning) is more expensive than prediction. The time cost for fine-tuning the sLLMs ranges from 15 seconds to 10 minutes, depending on the datasets used. In terms of prediction time, all approaches make predictions for up to hundreds of text units (documents) within seconds. The sLLMs cost less than 50% of Senti4SD and Stanford CoreNLP to make predictions, but cost two times more than the time needed by SentiCR, SentiStrength, and SentiStrength-SE.

2.1.4 Discussion

This section presents the lessons learned from our experiments and discusses threats to validity.

Lessons Learned

Fine-tuning sLLMs is promising for SA4SE. Lin et al. [99] mentioned that no prior SA4SE tool is ready for real usage of identifying sentiment expressed in SE data yet. We get similar results when applying the same approach (i.e., Stanford CoreNLP) to Stack Overflow posts (i.e., `StackOverflow` dataset). On the other hand, we found that even the worst-performing sLLM (i.e., ALBERT) achieves 0.66 in terms of macro-averaged F1-score, which outperforms Stanford CoreNLP by 27%. The micro-averaged F1-scores

[9] In our study, we utilized the original implementation of Senti4SD (https://github.com/collab-uniba/Senti4SD/). An alternative implementation (https://github.com/collab-uniba/pySenti4SD/) might offer enhanced efficiency. Evaluating and comparing these implementations remains an area for future research.

produced by the sLLMs range from 0.86 to 0.90. The promising effectiveness of sLLM-based approaches has also been observed on the other five datasets. Although there is no gold standard or concrete thresholds of various evaluation metrics to decide whether a SA4SE tool can be put into *real use*, our experiment results show that the sLLM-based approach is more ready than the existing techniques for SA in SE. Thus, we encourage researchers to consider simply fine-tuning pre-trained sLLM-based approaches as the baseline in future work. Moreover, we advocate inventing more advanced sLLM-based models to make SE-specific SA tools more practical.

Specific training (or fine-tuning) can boost performance. The approaches can be divided into two groups based on whether they are trained (or fine-tuned) on specific datasets or not. We fine-tuned all the sLLMs and trained SentiCR for each dataset. Based on our results, we found that all the approaches that have been trained (or fine-tuned) on SE datasets outperform those that have not been trained (or fine-tuned) across all six datasets. Moreover, we find that Senti4SD, which is designed based on Stack Overflow data [42], performs the best for `API` and `StackOverflow` datasets. These indicate that a tool trained on the same data source can perform better for the same or similar data sources.

Challenges in assigning sentiment labels. Previous work [99] shows that even human raters have more than 18% disagreements on the same sentences as sentiment identification may be subjective. We also observed that it is hard to determine the sentiment labels of some sentences. For example, the sentence "*It's always sad to see a reference like that go, but it was probably a good move.*" is labeled as *negative*. However, part of it (i.e., "it was probably a good move") should be considered as positive. Thus, there may be a need to introduce additional labels, e.g., mixed sentiment, or to go more fine-grained (i.e., attaching sentiment labels to phrases instead of sentences). Customized solutions may boost performance further. We also found that no approach can always achieve the best performance on all six datasets. It indicates that customization of the technical design is

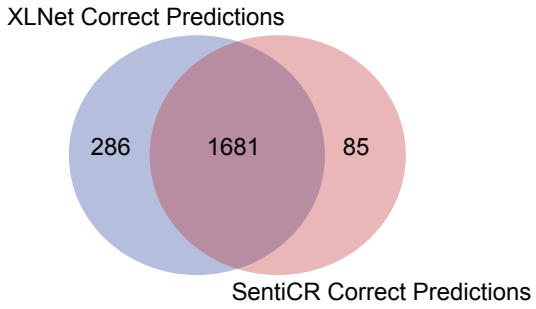


Figure 2.1: The Venn diagram of the correct predictions made by XLNet and SentiCR

also required in future work. It would be interesting to extend the current sLLM-based models to consider the specific properties of the SE datasets that we have.

Composition of different solutions may boost performance further. We find that some sentences can only be assigned correct sentiment labels by the (generally) underperforming SA4SE solution. To illustrate this, we conducted a brief error analysis on the largest `GitHub` dataset to help understand the different performances of different approaches. In total, we have 2,137 sentences in the test set. We focus on comparing the best performing approaches from the PRIOR and sLLM groups, i.e., SentiCR and XLNet. Figure 2.1 depicts the correct predictions produced by the two approaches. Among all the 2,137 sentences, XLNet and SentiCR correctly predict 1,967 (92%) and 1,766 (82%) sentences; among them, 1,681 (78%) are in common. From the Venn diagram, we find that although SentiCR performs worse (in general) than XLNet, for 85 sentences, it outperforms SentiCR. Table VIII shows some examples where one of the approaches fails, but the other is successful. Thus, by composing many different tools, we can boost the performance further. As future work, we want to explore the possibility of combining all the existing solutions for higher accuracy. Table 2.6 shows some examples of sentences from the `GitHub` dataset with the prediction results produced by XLNet and SentiCR. For each example, one of the two approaches makes a wrong prediction.

Table 2.6: Prediction Examples

Sentence	Label	XLNet	SentiCR
Yes, it would be really cool if you could update the wiki. But don't forget to say it will only work from version 1.4.0 forward!"	Positive	Positive	Negative
Thanks for your comments and tests! (Aptana is driving me nuts, I'm currently searching for another IDE)	Positive	Negative	Positive
Looks good. Mind if I add a CityHash implementation in here?	Neutral	Positive	Neutral
Strange indentation here	Neutral	Negative	Neutral
Pretty simple script for a TBC boss I say. I wonder who did it originally...	Negative	Neutral	Negative
If you mean #any_instance, you were better off not knowing. It's a nasty code smell.	Negative	Negative	Neutral

Threats to Validity

Threats to internal validity in our study relate to errors we may have made in our experiments and the quality of the manually labeled datasets utilized. We have released a replication package[10] for others to check and extend. Regarding the datasets, our research relies on publicly available datasets previously used in other studies. Consequently, while these datasets are integral to our analysis, we acknowledge that we inherit any inherent quality limitations from these prior works. This inherited limitation represents a potential constraint on our findings, as the original dataset quality has not been independently verified in our study.

Threats to external validity are related to the generalizability of our research and experiments. We consider six sentiment classification datasets, larger than those considered in a closely related work [99]. These six datasets are diverse in several aspects, e.g., scale, type of software artifacts, class distribution, etc. Our experimental setting follows Novielli et

[10] https://github.com/soarsmu/SA4SE

al. [123], for each dataset, uses 70% for training (or fine-tuning) and 30% for testing. In the future, we plan to employ k-fold cross-validation, a more rigorous evaluation method.

The threat to construct validity in our study arises primarily from the suitability of our evaluation metrics. In assessing SA4SE solutions, metrics such as precision, recall, and F1-score are commonly employed, as evidenced by their widespread use in the relevant literature (e.g., [123, 161, 121]).

2.2 How sLLMs Compare to bLLMs?

In the prior section, we show that sLLMs can outperform prior SA4SE tools. However, several challenges persist in the field of SA4SE with sLLMs. First, the accuracy of sLLMs can degrade when there is a lack of labeled data for fine-tuning. For instance, the Google Play dataset, which contains APP review comments, only has 341 labeled documents; among them, 25 are neutral. The fine-tuned sLLMs predicted none of the data points in the test set as neutral. While acquiring more labeled data can help mitigate this issue, manually labeling large volumes of data is time-consuming. The second challenge relates to the limitations of fine-tuning itself. Fine-tuning sLLMs requires updating some of the model parameters with domain-specific data. Lastly, the third challenge occurs in cross-platform settings [121], where SA4SE tools tend to perform poorly. Models trained from one dataset may not generalize well when tested on a different dataset, hindering the generalizability and effectiveness of existing SA4SE tools. Given these challenges, there is a need to explore more effective solutions for SA4SE.

Recently, bLLMs have shown promising results in many areas, spanning from general NLP tasks to specialized applications like software development. bLLMs are usually trained on massive corpora of texts and contain many parameters. For instance, GPT-

3 [37] is trained on 175 billion parameters. Llama [158] is trained on trillions of tokens and contains 7B to 65B parameters. They have drastically reduced the domain-specific training examples required for a particular application [50]. This adaptability has been a game-changer in reducing the need for domain-specific training data, as bLLMs can leverage their pre-existing knowledge to excel in diverse applications. bLLMs can make predictions conditioned on a few input-output examples without updating any model parameters and achieve success in various tasks [37, 50]. Nevertheless, their performance in SA4SE remains largely unexplored. The intriguing prospect of adopting bLLMs in this context lies in their ability to potentially address the challenges associated with fine-tuning sLLMs and the limitations observed in cross-platform settings.

To fill this gap, our work embarks on a journey to explore the effectiveness of bLLMs for SA4SE. To investigate the effectiveness of bLLMs for SA4SE, we conducted a comprehensive empirical study on five existing SE datasets. We first evaluate bLLMs under zero-shot and few-shot settings. For the zero-shot setting, we experimented with three different prompt templates. For the few-shot setting, we experimented with 1-, 3-, and 5-shot. The experimental results demonstrate that bLLMs can perform well under a zero-shot setting, while few-shot learning can further boost the performance. However, adding more shots does not guarantee an improvement in the performance. We also compared prompting bLLMs with fine-tuning sLLMs. We find that, on a dataset that lacks training data and the data is highly imbalanced, bLLMs can surpass sLLMs by a large gap. For the datasets that contain sufficient training data and more balanced data, sLLMs may still be preferred.

2.2.1 Experimented Language Models

bLLMs. We include three recently proposed bLLMs based on their performance in the MMLU benchmark on the chatbot leaderboard [11] in August 2023; the model name in the parenthesis is the exact model variant we used on the Hugging Face platform [168].

- **Llama 2-Chat** (`meta-llama/Llama-2-13b-chat-hf`) [159] is a fine-tuned version of Llama 2 that is optimized for dialogue use cases. Llama 2 uses the standard Transformer architecture [163] and it applies pre-normalization with RMSNorm [181], the SwiGLU activation function [148], and rotary positional embeddings [150]. Llama 2 made several improvements over Llama 1, including but not limited to more robust data cleaning, trained on 40% more total tokens, and doubled the context length.

- **Vicuna** (`lmsys/vicuna-13b-v1.5`) [49] is a chatbot trained by fine-tuning Llama 2 on 70K user-shared ChatGPT conversations. To better handle multi-turn conversations and long sequences, Vicuna is trained with the enhanced training script from Alpaca [155].

- **WizardLM** (`WizardLM/WizardLM-13B-V1.2`) [172] is another fine-tuned version of Llama 2. The authors propose *Evol-Instruct*, a novel method using bLLMs instead of humans to automatically mass-produce open-domain instructions of various complexity levels to improve the performance of bLLMs. The resulting bLLMs by fine-tuning Llama 2 with the evolved instructions is called WizardLM.

sLLMs. We include all the four sLLMs described in Section 2.1.1, i.e., BERT [55], RoBERTa [107], ALBERT [92], XLNet [175]. In addition, we also include a lightweight and memory-efficient variant of BERT, i.e., DistilBERT. We briefly describe DistilBERT

[11] https://huggingface.co/spaces/lmsys/chatbot-arena-leaderboard

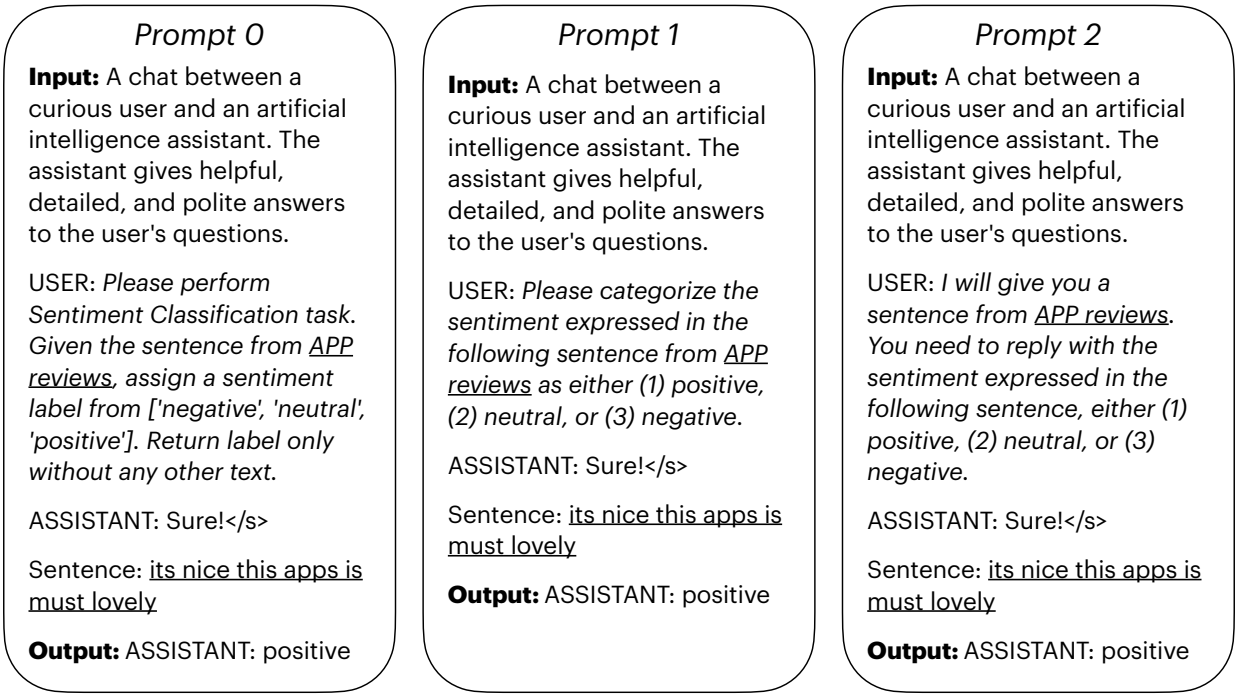


Figure 2.2: Zero-shot prompt template used by Vicuna and WizardLM.

here. Section 2.1.1 contains the description of the other four sLLMs. These sLLMs mainly differ in the pre-training tasks adopted. We also present the exact model we used on the Hugging Face platform [168] in the parenthesis.

DistilBERT (`distilbert-base-uncased`) [144] is a distilled and smaller version of the BERT model. DistilBERT is designed to be faster and more memory-efficient. DistilBERT adopts model compression or knowledge distillation to learn from a teacher BERT to capture the same knowledge but with fewer parameters. As DistilBERT combines efficiency and strong performance, it has been popular in research and industry settings.

2.2.2 Study Setup

Research Questions

In this work, we plan to answer the following research questions (RQs):

RQ1: *How do various prompts affect the performance of bLLMs in zero-shot learning for the SA4SE task?*

In this RQ, our initial focus is exploring the zero-shot learning scenario, where bLLMs are prompted without providing any labeled data. Prior studies have unveiled that bLLMs exhibit varying results even when prompted with semantically similar queries [129, 110]. Additionally, certain research findings have emphasized the substantial impact of different word orders within the prompt templates on the predictions [115, 53]. Given the straightforward nature of SA, our objective is to formulate equally straightforward prompts.

These prompts encompass two key components: the *Task Description* and the *Output Format*. The *Task Description* serves the purpose of elucidating the task clearly and concisely. In our specific context, the task pertains to SA, and we articulate it through various expressions within the three templates. Importantly, the sentence origin (e.g., from APP reviews, from Stack Overflow) remains consistent across all prompt templates, enabling the distinction of diverse contexts and domains.

The *Output Format* component is designed to provide bLLMs with guidance for generating responses in a specific format, facilitating the sentiment label extraction. To maintain generality, we employ an identical prompt template for all five datasets.[12] Figure 2.2 shows the three prompt templates we used for zero-shot setting. While all prompt templates share a semantic similarity, they differ in their syntactic structure. Our inspiration for the first prompt template (i.e., Prompt 0) draws from Zhang et al. [188]. They designed the prompt to include only essential components, namely the task name, task definition, and output format. For Prompt 1 and Prompt 2, we introduce slight variations in expression.

[12]There is only a minor difference in the `Jira` dataset, given it only contains two sentiments, i.e., negative and positive. We reduced the scope to only two options in the templates used by `Jira`.

Input: <s>[INST] <<SYS>>

A chat between a curious user and an artificial intelligence assistant. The assistant gives helpful, detailed, and polite answers to the user's questions.

<</SYS>>

Please perform Sentiment Classification task. Given the sentence from APP reviews, assign a sentiment label from ['negative', 'neutral', 'positive']. Return label only without any other text.

Example Sentence: this app is a waste of time. i can log in but it always freezes and eventually crashes. i do not feel confident with the security on this app if wf can leave us hanging with a horrible app for such a big bank. uninstalled htc one

Label: negative

Demonstration

[/INST]

Sure!

</s><s>[INST]

Sentence: good it's good app

Label:

[/INST]

Output: Label: positive

Figure 2.3: Few-shot prompt template (with $k = 1$) utilized by Llama 2-Chat.

RQ2: *How do various shots affect the performance of bLLMs in few-shot learning for the SA4SE task?*

In the context of few-shot learning, we leverage the best-performing zero-shot prompt template, namely *Prompt 0*. We enrich *Prompt 0* with various numbers of examples filled in the **Demonstration** part. The demonstration part encompasses k ($k = 1, 3, 5$) examples and corresponding ground-truth labels, adhering to the desired format.

Figure 2.3 illustrates the few-shot prompt template employed by Llama 2-Chat on the `GooglePlay` dataset. In the depicted figure, the demonstration segment (enclosed within the red box) comprises only one example. In the case of a 3-shot or 5-shot setup, this demonstration section would encompass a greater number of example sentences and their corresponding gold labels. We systematically sampled 1-, 3-, and 5-examples from the training data of each dataset, subsequently populating the demonstration segment within the template. This approach ensures that under the k-shot setting, different bLLMs receive

Table 2.7: Dataset statistics. Neg. stands for negative, Neu. stands for neutral, Pos. stands for positive, and Non-neg. stands for non-negative.

Dataset	Total	Test	Sampled Test	Avg. Tokens
Gerrit	1,600: 398 (Neg.) 1,202 (Non-neg.)	160	114	29
GitHub	7,122: 2,087 (Neg.) 3,022 (Neu.) 2,013 (Pos.)	713	250	19
GooglePlay	341: 130 (Neg.) 25 (Neu.) 186 (Pos.)	35	33	27
Jira	926: 636 (Neg.) 290 (Pos.)	93	76	9
StackOverflow	1,500: 178 (Neg.) 1,191 (Neu.) 131 (Pos.)	150	109	11

the same set of examples.

RQ3: *How do bLLMs compare with fine-tuned sLLMs on the SA4SE task?*

We compare the best macro-F1 and micro-F1 obtained by prompting bLLMs and the fine-tuned sLLMs.

Dataset

In this work, we experimented with the existing manually labeled datasets from five distinct platforms: Gerrit, GitHub, Google Play, Jira, and Stack Overflow. The description of these five datasets is available in Section 2.1.2.

We split each dataset with a ratio of 8:1:1, which stands for training, validation, and test, respectively. We did a stratified split where we kept the original class distribution in training, validation, and test. Since running bLLMs is expensive, they usually contain billions of parameters, we did a sampling on all the test data with a confidence level of 95% and a margin of error of +/- 5%.[13] Similarly, we also kept the class distribution the same as in the original whole dataset.

[13]We included all the provided test data in the `GooglePlay` dataset, as the number of sampled data is only 2 data points fewer than the whole test data.

Table 2.7 presents the statistics of investigated datasets, specifically the average number of tokens per document. Notably, `Gerrit` and `GooglePlay` exhibit longer text, likely due to the inclusion of code review and APP review comments, which often span multiple sentences. Although the `GitHub` dataset also comprises pull request and commit comments, they are typically short in nature. Conversely, the `StackOverflow` dataset and `Jira` dataset include *sentences* from Stack Overflow and Jira, respectively. While some document units in these datasets contain multiple sentences and others just one, we collectively refer to them as *documents*.

Evaluation Metrics

Following the prior works, we also use macro- and micro-averaged precision, recall, and F1-score. Section 2.1.2 details the formula for calculating these metrics.

We report both the macro-F1 and micro-F1 scores as they show the balance between precision and recall. We attach the full result in our replication package.[14] From these formulas, we can find that micro-F1 emphasizes overall accuracy, while macro-F1 gives equal weight to each class's performance. As we do not give weights to the classes and want to consider the F1 in each class, we choose macro-F1 as the main metric. This choice of preferring macro-F1 also aligns with the prior work [121].

Implementation Details

We run bLLMs and sLLMs on a machine with four NVIDIA RTX A5000. For bLLMs, we run each model with four GPUs. For sLLMs, we fine-tune each model with one GPU.

[14] https://github.com/soarsmu/LLM4SA4SE

Prompting bLLMs. We use heuristics to extract the sentiment returned by bLLMs. When bLLMs consider it hard to decide the sentiment (they replied with a sentiment other than the three polarities, e.g., "mixed"), we label the predicted sentiment as "neutral". As there is no "neutral" sentiment in the `Jira` dataset, if any LLM predicts the sentiment as "neutral", we label the predicted sentiment as the opposite of the ground-truth label. In the few-shot setting, we select k examples ("shots") at random from the training set ($(k = \{1, 3, 5\})$), and for each example, we append its ground-truth label. To make a new prediction for a new example, we append one sentence from the test set.

Fine-tuning sLLMs. We fine-tuned all the sLLMs with the training data. For each epoch, we calculate their macro-F1 score on the validation data. We fine-tune each sLLM 5 epochs. We save the best-performing model, i.e., achieving the highest macro-F1 score on the validation data, as the final model. We then evaluate the best model on the test data. We used the following sets of hyper-parameters for all the sLLM: learning rate of 2e-5, batch size of 32, and max length of 256.

2.2.3 Study Results

RQ1: Impact of different prompts on the performance of bLLMs with zero-shot learning

It is worth noting that Vicuna and WizardLM adopts the same style of prompt template; while Llama 2-Chat employs a different prompt template in pre-training, resulting in slight differences in prompt formats[15] (We show an example of Llama 2-Chat prompt in Figure 2.3, where the zero-shot template is the same template excluding the "Demonstration" part).

[15] https://huggingface.co/blog/llama2#how-to-prompt-llama-2

Table 2.8: Zero-shot Performance: Comparative Results of LLMs Across Five Datasets. Cells highlighted in red indicate the highest scores achieved among the three prompts executed by each respective model.

	Model	*Variant*	*Gerrit*	*GitHub*	*GooglePlay*	*Jira*	*StackOverflow*
Macro-F1	**Vicuna**	**0**	0.73	0.72	0.98	0.85	0.59
		1	0.73	0.65	0.74	0.69	0.56
		2	0.7	0.67	0.82	0.75	0.53
	WizardLM	**0**	0.69	0.71	0.8	0.81	0.41
		1	0.69	0.7	0.82	0.82	0.59
		2	0.68	0.7	0.79	0.77	0.52
	Llama 2-Chat	**0**	0.73	0.68	0.89	0.83	0.45
		1	0.71	0.64	0.89	0.71	0.5
		2	0.75	0.68	0.89	0.78	0.51
Micro-F1	**Vicuna**	**0**	0.81	0.72	0.97	0.86	0.78
		1	0.82	0.66	0.8	0.71	0.82
		2	0.82	0.67	0.89	0.76	0.78
	WizardLM	**0**	0.8	0.71	0.86	0.82	0.65
		1	0.8	0.7	0.89	0.83	0.73
		2	0.79	0.7	0.89	0.78	0.67
	Llama 2-Chat	**0**	0.82	0.68	0.91	0.84	0.61
		1	0.82	0.64	0.91	0.71	0.72
		2	0.83	0.68	0.94	0.79	0.64

Table 2.8 presents the outcomes obtained from our investigation into three distinct bLLMs using three distinct zero-shot prompts. Notably, we observe varying performance levels among these bLLMs when employed with different prompt templates. Furthermore, it is noteworthy that even when using the same LLM, the optimal prompt template can vary depending on the dataset under consideration.

Specifically, regarding the macro-F1 score, *Prompt 0* emerges as the most effective choice, yielding the highest scores in 17 instances. Following closely, *Prompt 1* leads in 13 instances. Interestingly, *Prompt 2*, while achieving the top performance on only 9 occasions, occasionally surpasses *Prompt 1* by a significant margin, notably in the case of the Llama 2-Chat within the `StackOverflow` dataset. Regarding the micro-F1 scores, both *Prompt 0* and *Prompt 1* achieved the highest scores 7 times, while *Prompt 2* ranked the first 5

Table 2.9: Zero-shot: Score difference between the highest and lowest ones with each LLM on one dataset and the value in the parenthesis shows the difference percentage.

	Model	*Gerrit*	*GitHub*	*GooglePlay*	*Jira*	*StackOverflow*
Macro-F1	**Vicuna**	0.03 (4.3%)	0.07 (10.8%)	0.24 (32.4%)	0.16 (23.2%)	0.06 (11.3%)
	WizardLM	0.01 (1.5%)	0.01 (1.4%)	0.03 (3.8%)	0.05 (6.5%)	0.18 (43.9%)
	Llama 2-Chat	0.04 (5.6%)	0.04 (6.3%)	0	0.12 (16.9%)	0.06 (13.3%)
	Avg. Diff	3.80%	6.17%	12.07%	15.53%	22.83%
Micro-F1	**Vicuna**	0.01 (2.1%)	0.06 (12.5%)	0.17 (35.4%)	0.15 (31.3%)	0.04 (8.3%)
	WizardLM	0.01 (1.3%)	0.01 (1.4%)	0.03 (3.5%)	0.05 (6.4%)	0.08 (12.3%)
	Llama 2-Chat	0.01 (1.2%)	0.04 (6.3%)	0.03 (3.3%)	0.13 (18.3%)	0.11 (18.0%)
	Avg. Diff	1.5%	6.7%	14.0%	18.7%	12.9%

times. These results show that Prompt 0 can achieve overall best results considering both macro-F1 and micro-F1.

Table 2.9 provides an insight into the variance within each result group under the zero-shot setting. We define a result *group* as results generated by the same LLM when applied to the same dataset with varying prompts. This analysis aims to underscore the impact of prompt selection on performance. Remarkably, within the same group, we observe disparities as substantial as 43.9%. Expanding our examination to encompass different models operating on identical datasets reveals an average difference as substantial as 22.83%. This discovery underscores the sensitivity of bLLMs to the choice of prompts in zero-shot learning.

Answer to RQ1: In the SA4SE context, it is evident that bLLMs exhibit sensitivity to prompts in zero-shot learning scenarios. When employing various prompt templates, the average macro-F1 score difference spans from 3.8% to 22.83%, and the average micro-F1 score difference ranges from 1.5% to 18.7%.

Table 2.10: Few-shot Performance: Comparative Results of LLMs Across Five Datasets. Cells highlighted in red indicate the highest scores achieved among the three prompts executed by each respective model.

	Model	*Shot*	*Gerrit*	*GitHub*	*GooglePlay*	*Jira*	*StackOverflow*
Macro-F1	**Vicuna**	1	0.74	0.68	0.74	0.77	0.56
		3	0.73	0.72	0.82	0.86	0.65
		5	0.71	0.72	0.77	0.89	0.64
	WizardLM	1	0.76	0.68	0.89	0.78	0.54
		3	0.75	0.72	0.87	0.9	0.59
		5	0.75	0.71	0.82	0.91	0.54
	Llama 2-Chat	1	0.69	0.54	0.89	0.82	0.42
		3	0.69	0.6	0.87	0.84	0.46
		5	0.68	0.61	1	0.89	0.47
Micro-F1	**Vicuna**	1	0.82	0.69	0.8	0.78	0.78
		3	0.81	0.72	0.89	0.87	0.83
		5	0.78	0.72	0.83	0.89	0.82
	WizardLM	1	0.83	0.67	0.94	0.79	0.67
		3	0.84	0.72	0.94	0.91	0.74
		5	0.83	0.7	0.91	0.92	0.74
	Llama 2-Chat	1	0.78	0.54	0.94	0.83	0.42
		3	0.79	0.6	0.94	0.86	0.51
		5	0.77	0.6	1	0.89	0.59

RQ2: Impact of different shots on the performance of bLLMs with few-shot learning

Table 2.10 showcases the outcomes of few-shot learning utilizing three distinct bLLMs across five diverse datasets. It is important to reiterate that a result "group" signifies the results produced by the same LLM when applied to the same dataset with varying shot numbers.

In summary, when considering macro-F1, the 5-shot configuration emerges as the leader in seven instances, followed by the 3-shot setup in five cases, and the 1-shot configuration excels in four instances. Regarding micro-F1, the 5-shot configuration is at the top seven times, the 3-shot setup prevails nine times, and the 1-shot configuration leads twice. Generally, the trend for both macro-F1 and micro-F1 indicates that having more than one

example is more beneficial than having only one.

Among the three models, Llama 2-Chat consistently benefits from having more examples, except for the `Gerrit` dataset concerning macro-F1 and micro-F1. In the case of Vicuna and WizardLM, the impact of additional examples is noticeable primarily on the `Jira` dataset, affecting both macro-F1 and micro-F1. This underscores the fact that the influence of additional examples can vary depending on the LLM and dataset employed. This can be explained by recognizing that SA, especially the sentiment classification task central to our work, is relatively straightforward. Consequently, the effects of increasing training examples might not be as pronounced. Given all the bLLMs benefit from more shots on the `Jira` dataset, we investigate the potential reason. One unique characteristic of the `Jira` dataset is that it is a binary-class dataset. Despite specifying the available label list as [*'negative'*, *'positive'*] in the prompt templates, bLLMs tend to predict the sentiment as *neutral*. We examined the cases where bLLMs predicted the sentiment as *neutral*:

In the zero-shot setting, Vicuna explicitly predicts the *neutral* label 4, 19, and 9 times with *Prompt 0, 1, 2*, respectively. Similarly, Llama 2-Chat deviated from the template to some extent but performed better than Vicuna, predicting the *neutral* label 1, 14, and 5 times. WizardLM, on the other hand, demonstrated a stronger adherence to the template by predicting *neutral* only 1, 0, and 1 times in the same scenarios.

This issue was mitigated when we introduced more shots under the few-shot setting: In the case of Vicuna, it explicitly predicted *neutral* 11, 8, and 5 times with 1-shot, 3-shot, and 5-shot prompt templates, respectively. Like zero-shot learning, WizardLM also predicted *neutral* 1, 0, and 1 times. The occurrences of Llama 2-Chat predicting the *neutral* label under few-shot learning were comparatively lower. With 1-shot, 3-shot, and 5-shot prompt templates, the numbers were 4, 4, and 3, respectively. This shows the potential benefits brought by introducing more shots.

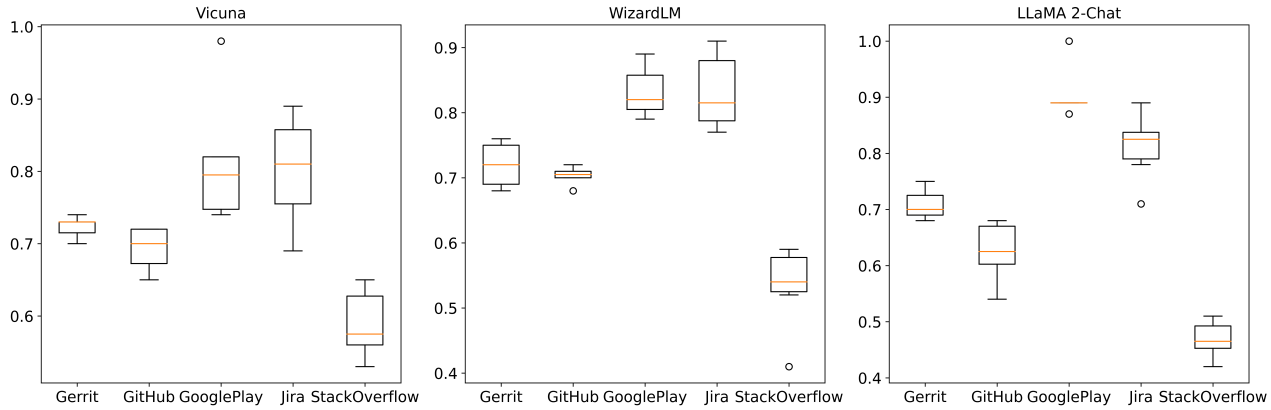


Figure 2.4: Sensitivity of different prompt designs. Four different prompts give the performance variance of each dataset. The circles depicted in the figure represent outlier data points.

Other than benefits brought by more shots, we also notice the decline in macro-F1 with an increase in the number of examples is apparent, particularly when applying all three bLLMs to the `Gerrit` dataset and when using WizardLM on the `GooglePlay` dataset. One plausible explanation for this phenomenon is that an increased number of examples leads to longer prompts, which could potentially confuse the bLLMs. As indicated by Table 2.7, the documents from the `Gerrit` dataset have the highest average number of tokens. Consequently, introducing more examples results in even lengthier prompts. This observation aligns with prior research on bLLMs in the broader SA domain. [188].

Figure 2.4 provides a clearer illustration of this phenomenon. The box plot delves into the variance of macro-F1 scores achieved by different prompts for each model on each dataset. All six prompts examined in their study are considered. This figure reveals that the influence of different prompts on performance varies depending on both the model and the dataset. In general, the models demonstrate differing levels of sensitivity to prompts. Notably, on the `Jira` dataset, all models exhibit high sensitivity to prompts, signifying that the choice of prompt has a substantial impact on results. In contrast, on the `Gerrit` and `GitHub` datasets, models appear less responsive to different prompts, suggesting that the choice of prompt has a relatively smaller effect on their performance.

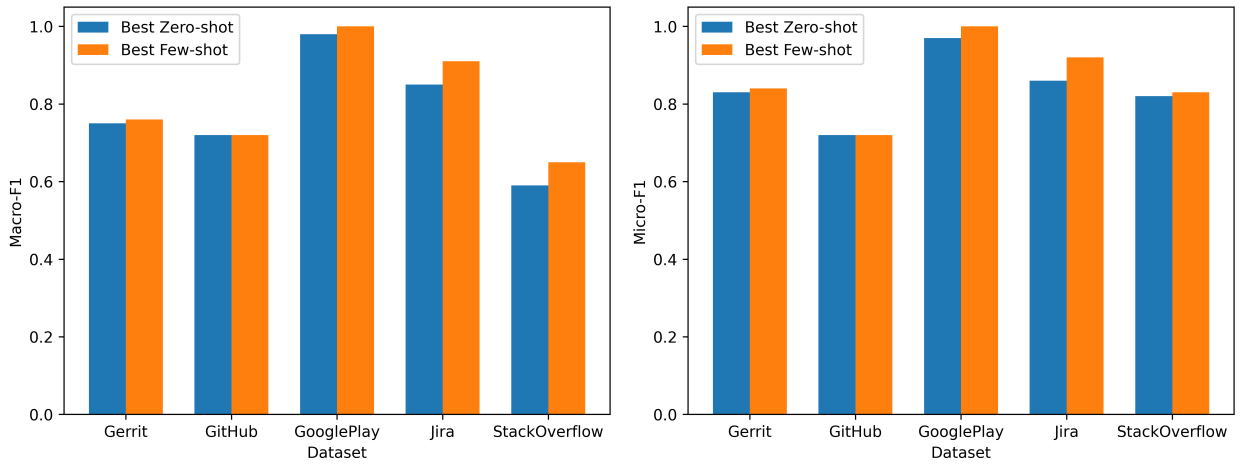


Figure 2.5: Comparison of the highest macro-F1 and micro-F1 scores achieved through zero-shot learning and few-shot learning.

We also compare the best results of any of the three bLLMs under few-shot learning and those achieved under zero-shot learning. Figure 2.5 illustrates that, in all the five datasets, the highest macro-F1 score achieved through few-shot learning either surpasses or equals the highest macro-F1 score attained via zero-shot learning. We can observe a similar trend in terms of the micro-F1 scores. This trend is particularly pronounced on the `Jira` dataset, where bLLMs perform notably better under the few-shot learning paradigm. We conducted a Wilcoxon signed-rank test on two pairs of comparisons: the best zero-shot versus few-shot performance in terms of macro-F1 and the best zero-shot versus few-shot performance in terms of micro-F1. Notably, both comparisons yielded p-values of 0.07. Hence, it is important to recognize that while few-shot learning does demonstrate superior results, the margin of improvement is not substantial.

Answer to RQ2: Although the top-performing bLLM achieved equal or higher macro- and micro-F1 scores in all five datasets with few-shot learning, the difference between the zero-shot learning was insignificant. In addition, there is no guarantee that the same bLLM will exhibit improved performance through few-shot learning over zero-shot learning.

RQ3: Comparison between fine-tuned sLLMs and bLLMs.

Table 2.11: Results of LLMs compared with fine-tuned SLMs. Cells highlighted in red indicate the highest scores achieved among the three prompts executed by each respective model.

	Model	*Gerrit*	*GitHub*	*GooglePlay*	*Jira*	*StackOverflow*
Macro-F1	**Vicuna**	0.74	0.72	0.98	0.89	0.65
	WizardLM	0.76	0.72	0.89	0.91	0.59
	Llama 2-Chat	0.75	0.68	1	0.89	0.51
	ALBERT	0.73	0.9	0.56	0.97	0.64
	BERT	0.75	0.92	0.49	0.95	0.57
	DistilBERT	0.81	0.92	0.57	0.95	0.6
	RoBERTa	0.74	0.94	0.42	0.95	0.68
	XLNet	0.77	0.91	0.39	0.94	0.67
Micro-F1	**Vicuna**	0.82	0.72	0.97	0.89	0.83
	WizardLM	0.84	0.72	0.94	0.92	0.74
	Llama 2-Chat	0.83	0.68	1	0.89	0.72
	ALBERT	0.81	0.9	0.8	0.97	0.84
	BERT	0.8	0.92	0.71	0.96	0.84
	DistilBERT	0.86	0.92	0.83	0.96	0.84
	RoBERTa	0.81	0.94	0.63	0.95	0.86
	XLNet	0.83	0.91	0.57	0.95	0.89

Table 2.11 presents a comparative analysis of the top-performing results obtained through two distinct approaches: prompting bLLMs and fine-tuning sLLMs.

On the `GooglePlay` dataset, where the fine-tuning data is limited to fewer than 300 data points and with a negative:neutral:positive ratio of 26:5:37, bLLMs demonstrate a remarkable performance advantage. The most effective LLM, Vicuna, significantly enhances the performance of the leading sLLM, DistilBERT, by a substantial improvement of 71.9%. It indicates that when labeled data is scarce and the training dataset is highly imbalanced, fine-tuning bLLMs should be the preferred approach over sLLMs.

On the contrary, we have observed that fine-tuning sLLMs produces superior results on the `GitHub` and `Jira` datasets. The `GitHub` dataset benefits from a larger training

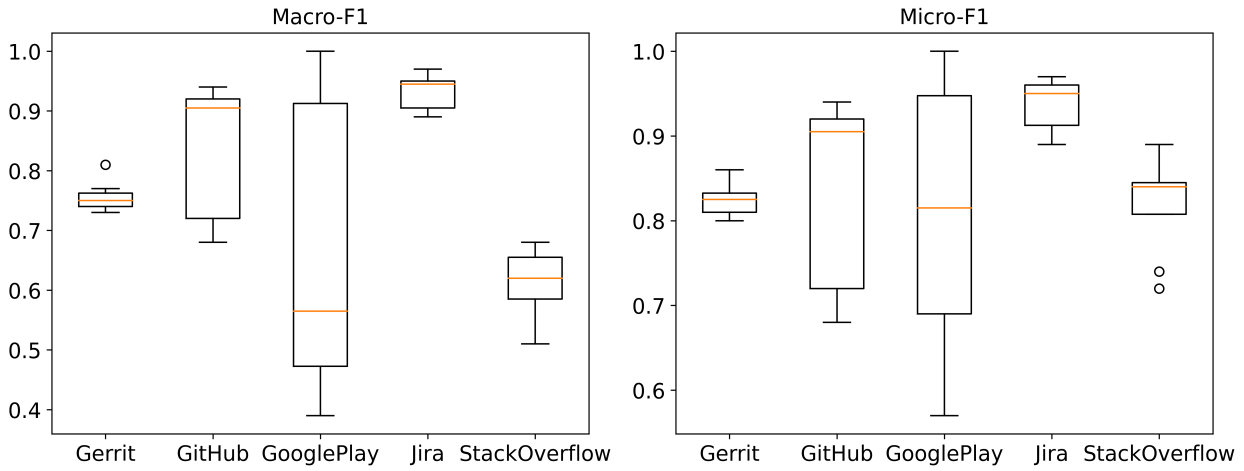


Figure 2.6: Performance variance of all the models on each dataset.

dataset and a more evenly distributed class structure, making it particularly well-suited for fine-tuning sLLMs. In contrast, the `Jira` dataset, though smaller than `GitHub`, offers a more favorable class distribution than `GooglePlay`, with a negative-to-positive ratio of approximately 1:2.

For the `Gerrit` and `StackOverflow` dataset, both bLLMs and sLLMs exhibit relatively modest performance. The best-performing LLM achieves a macro-F1 score of 0.65, while the top sLLM achieves a macro-F1 score of 0.68 on the `StackOverflow` dataset. The corresponding score is 0.76 and 0.81 on the `Gerrit` dataset. Both datasets share the challenge of imbalanced label distribution, which explains the limited success of sLLMs. Furthermore, for the `StackOverflow` dataset, given the short length of the sentences in this dataset, neither bLLMs nor sLLMs may have sufficient contextual information to make accurate sentiment predictions.

Moreover, in Figure 2.6, we observe the performance of all the models across different datasets. Notably, on the `GooglePlay` dataset, there is a significant variance, with bLLMs standing out by achieving the highest macro-F1 and micro-F1 scores. This underscores bLLMs' superiority over sLLMs on this specific dataset.

On the other hand, for the `Gerrit`, `Jira`, and `StackOverflow` datasets, the variance is comparatively smaller, suggesting that while sLLMs outperform bLLMs, the margin of difference is not substantial.

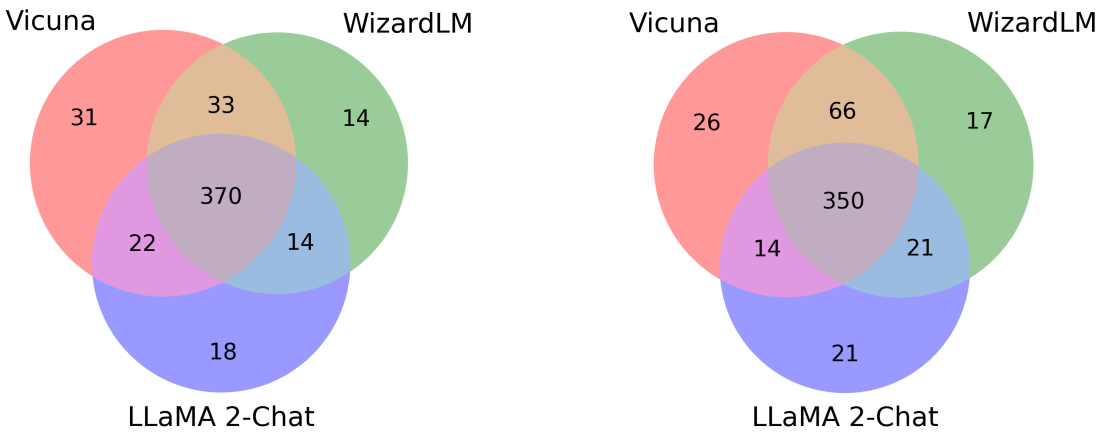
Conversely, in the case of the `GitHub` dataset, sLLMs demonstrate a substantial advantage over bLLMs. This discrepancy is likely due to the abundance of training data available for the `GitHub` dataset. It reinforces that sLLMs are most effective when ample, balanced training data is available.

Answer to RQ3: In scenarios with limited labeled data and pronounced class imbalance, prompting bLLMs is a more effective strategy, outperforming fine-tuning sLLMs, e.g., in the `GooglePlay` dataset, Llama 2-Chat outperforms the DistilBERT by 75.4%. In contrast, when ample training data is available and the dataset demonstrates a balanced distribution, the preference should lean toward sLLMs as the more suitable approach, e.g., in the `GitHub` dataset, RoBERTa outperforms Vicuna by 30.6%.

Error Analysis

In this section, we conducted a quantitative and qualitative analysis to understand the main cause of misclassification made by bLLMs. In the first part, we choose the results achieved by the best-performing templates for analysis, i.e., for zero-shot learning, we analyze the results by *Prompt 0*; for few-shot learning, we analyze the results by *5-shot* prompt.

In Figure 2.7, it is evident that all three bLLMs consistently generate accurate predictions in a substantial portion of instances. Specifically, in the realm of zero-shot learning, these models collectively predicted the correct sentiments in 73.7% cases, and in the context of few-shot learning, they made the correct prediction in 61.4% cases. Notably, both Vicuna and WizardLM stand out as the bLLMs with the highest degree of overlapping predictions



(a) Zero-shot learning with Prompt 0

(b) Few-shot learning with 5-shot prompt

Figure 2.7: The Venn diagram of the correct predictions made by bLLMs.

Table 2.12: Overlap in Misclassification Across LLMs in Zero-Shot and Few-Shot Settings. The *Common* Column Indicates Misclassifications Shared Between Both Settings.

	Misclassified (% of the test set)			***Test set size***
	Zero-shot	***Few-shot***	***Common***	
Gerrit	15 (13.2%)	10 (8.8%)	8 (7.0%)	114
GitHub	39 (15.6%)	41 (16.4%)	23 (9.2%)	250
GooglePlay	1 (2.9%)	0	0	35
Jira	8 (10.5%)	4 (5.3%)	4 (5.3%)	76
StackOverflow	19 (17.4%)	14 (12.8%)	11 (10.1%)	109
Total	82 (14%)	69 (11.8%)	46 (7.9%)	584

across both scenarios. In the zero-shot context, they share common predictions in an impressive 83.3% of their respective correct predictions, while in the few-shot scenario, this shared correct prediction rate remains substantial at 70.1%. Conversely, the lowest degree of common correct predictions is observed between Vicuna and Llama 2-Chat, with rates of 80.3% and 69.1% for zero-shot and few-shot scenarios, respectively. These results underscore the bLLMs' capacity to achieve comparable success rates in most cases while also emphasizing their unique strengths.

Now, we shift our focus to the errors made by these bLLMs. Table 2.12 demonstrates

Table 2.13: Distribution of error categories and their percentage among the whole error cases.

Error Category	**# Cases (%)**
Polar facts	16 (34.8%)
Subjectivity in annotation	13 (28.3%)
General error	8 (17.4%)
Politeness	5 (10.9%)
Implicit sentiment polarity	4 (8.7%)

again that overall, few-shot learning is more effective than zero-shot learning, as all the bLLMs misclassified more cases under zero-shot learning. However, we also notice that the number of common misclassification by both settings is 46, which accounts for 7.9% of the total test cases. To better understand the difficulties and challenges faced by bLLMs on the task of SA4SE, we manually examined these cases. In our experiments, all the bLLMs did not get any correct prediction no matter the setting. We adopt the error categories provided and used by Novielli et al. [123, 121]. Two authors (from here on, evaluators) first assigned a category to each document separately, which may not cover all the categories identified by Novielli et al. Afterward, they discussed and achieved agreement on the conflicts.

Table 2.13 shows the error categories of these common misclassifications by all the bLLMs. *Polar facts* emerge as the most prominent category, representing the majority of failure cases. In some cases, the sentence describes a fact, which may usually invoke for most people a positive or negative feeling, i.e. the annotator considered the described situation either as desirable or undesirable. They have been annotated inconsistently across different datasets. For example, in `GitHub` and `StackOverflow` datasets, they were labeled as *neutral*. However, in `Gerrit` and `Jira` datasets, they were labeled as *negative*. For instance, in the `Jira` dataset, *There is no need to reference $wizard, it's an object* was labeled as negative. In the `GitHub` dataset, *Ok, I'll fix them.* was labeled as neutral.

The second most prevalent category, *Subjectivity in annotation*, comprises cases where the evaluators' interpretation of sentiment differed from the originally assigned label. As recognized in prior works [102], sentiment or option itself is subjective. Similarly, sentiment annotation is also a subjective activity. Depending on personality traits or personal disposition, different annotators' perceptions of emotions might vary [145]. Thus, it is not rare that the evaluators have different perceptions from the original annotators. One instance is from the `StackOverflow` dataset, *Less likely to be blocked by paranoid firewall configurations.*. The evaluators consider this sentence as *positive*, while the ground-truth label is *negative*. Depending on which perspective we think, both make sense: we consider it positive, as we focus on "less likely to be blocked." However, the annotators may give more weight to "paranoid firewall configurations".

General errors account for 17.4% of cases and occur when the model fails to identify clues in the document that would be readily apparent to a human. For instance, emoticons can signal sentiment, as observed in the sentence from the `GitHub` dataset, "yep, it's work, but I need to add user and password for proxy connection=('." This sentence may convey negative sentiment to a human, particularly due to the emoticon "=(" embedded within it.

Politeness contributes to 10.9% of the error cases, arising when the presence of phrases like "thanks" or "sorry" leads to inconsistencies across different datasets. For instance, in the `GitHub` dataset, the sentence "sorry I did not realize you were already there..." was labeled as "negative", although some individuals may perceive it as "neutral". Similarly, in the `StackOverflow` dataset, "Good luck!" was labeled as 'neutral', but certain interpretations could classify it as 'positive'. These inconsistencies pose challenges for models when predicting labels with limited examples.

Lastly, *Implicit sentiment polarity* accounts for 8.7% error cases. When there is a lack of explicit sentiment clues, it could be hard to decide which sentiment is contained. For

instance, *Yes, it did not cause message loss just unnecessary retransmits.*, this sentence was annotated as negative, however, there is no obvious sentiment clue.

In summary, due to the inconsistency in labeling rules and the subjective nature of the task, challenges arise where bLLMs may struggle to improve significantly. However, in the case of general errors, there is a potential for improvement as bLLMs continue to advance.

2.2.4 Discussion

Implications for Future Research

Based on the experimental results in our study, we derive the empirical guidelines for future research on SA4SE and SE in general as follows.

Effective prompt engineering unlocks the full potential of bLLMs. Our experiments reveal a crucial insight: while prompt templates may appear similar at first glance, determining which one will yield the highest accuracy requires actual execution and template refinement. In this context, manual crafting of prompt templates proves exceptionally advantageous, particularly in zero-shot scenarios. It is through this meticulous process that we can fully harness the capabilities of bLLMs. In the case of few-shot learning, dependent on the characteristics of the task, it may not always be helpful to add more shots. The longer context can confuse bLLMs and lead to worse results.

Select your approach: considering the size and class distribution of the dataset. When determining the most suitable strategy for a specific task, it is important to consider the size and class distribution of the dataset. Based on our empirical results, it is crucial to recognize that, in cases with ample and balanced training data, fine-tuning sLLMs remains the preferred choice. This guideline is applicable to numerous SE tasks. If there are

already manually curated datasets available or acquiring labeled data is not a significant challenge, fine-tuning sLLMs represents a straightforward and effective option. However, in scenarios where labeled datasets are scarce, bLLMs emerge as a potential solution.

Additional guidelines for rule setting in human-labeled SA datasets or prompt design. As elaborated in Section 2.2.3, the inconsistency in labeling practices across various datasets poses a significant challenge for bLLMs to predict labels accurately. To enhance performance, we propose two potential approaches: 1. Encouraging human annotators to adhere to general labeling rules when annotating data. 2. Empowering bLLMs to incorporate dataset-specific labeling rules. Recall that despite restricting the label options to "positive" and "negative", bLLMs still occasionally return "neutral" labels when assessing the `Jira` dataset. This observation underscores the importance of further exploring and refining prompts to match the dataset characteristics.

Threats to Validity

Threats to Internal Validity. In our empirical study, a potential source of bias may arise from the choice of prompt templates. To mitigate this, we conducted experiments in the zero-shot setting using three different prompt templates. Additionally, we examined the influence of the number of shots in the few-shot setting. It is important to note that we based our prompt templates on prior work [188]. Another concern is the risk of data leakage. Nevertheless, the dataset we utilized cannot be directly accessed by visiting a webpage; it requires downloading from a specific URL. Consequently, the likelihood of data leakage is considered low. A further issue pertains to the quality of the labeled dataset. We did not generate new datasets but relied on pre-existing ones from other sources. Consequently, we inherit this quality concern from the original works. However, as described in Section 2.2.2, in the original labeling process, each dataset was labeled by

two or more labelers individually and resolved the conflict by involving another labeler. Thus, we consider the threat to be minimal. Lastly, our decision to categorize non-explicit responses from bLLMs as *neutral* poses a potential threat. This decision, aimed at ensuring consistency in SA, is based on the rationale that “mixed" sentiments typically indicate a balanced or uncertain position, closely aligning with a *neutral* stance.

Threats to External Validity. Our findings may not necessarily generalize to data from other platforms. Nevertheless, we have taken steps to mitigate this threat by considering data from five distinct platforms. It is important to recognize that our results are specific to the dataset and experimental setup we employed. In the few-shot learning setting, our results are contingent on randomly sampled examples. Nevertheless, our experiments and results still offer valuable insights, demonstrating that bLLMs can be a promising approach when dealing with a scarcity of annotated data. In the future, we plan to expand our analysis by incorporating additional datasets from various platforms and exploring more diverse prompt templates to enhance our understanding of leveraging bLLMs for SE in SE further.

2.3 Summary

In this work, we first conducted an extensive comparative study on the performance of prior SA4SE tools and smaller-size large language models (sLLMs). We are the first to investigate the effectiveness of various sLLMs for the SA4SE task. Our comparative study includes six datasets: GitHub pull-request and commit comments, API reviews from Stack Overflow, mobile APP reviews from Google Play, Stack Overflow posts, Jira issue comments, and code review comments. Our experimental results reveal that the best-performing fine-tuned sLLM outperforms the best-performing prior SA4SE tool by

6.5% to 35.6% in terms of the macro- and micro-averaged F1 scores. Overall, sLLM-based approaches are more ready to be applied in the real world for SA of SE data than the existing SA4SE tools.

In the follow-up work, we mark the initial step towards comprehending the potential of utilizing prompting bLLMs in discerning sentiment within SE domain documents. Our experiments reveal that in cases with limited annotated data, bLLMs outperform sLLMs, and zero-shot learning is a viable approach. However, when substantial and well-balanced training data is available, fine-tuning sLLMs is the preferable strategy over prompting bLLMs.

Chapter 3

Duplicate Bug Report Detection

This chapter presents two studies, in Section 3.1, we conduct a benchmark study and in Section 3.2, we propose a new approach.

3.1 Benchmarking Study

Despite the many research works and practitioners' adoption of DBRD, it is unclear which DBRD technique can recommend the duplicate BR most accurately overall. The most recent work by Rodrigues et al. [138] shows that `SABD` [138] outperforms `REP` [152] and `Siamese Pair` [54]. However, their experiments are only limited to a collection of old BRs from Bugzilla ITSs, in which the latest data used belongs to the year 2008. Concurrent with the work by Rodrigues et al. [138], Xiao et al. [170] and He et al. [75] have proposed other DBRD solutions. They have not been compared to each other. Besides, they did not compare with the tools used in practice. This motivates us to (1) create a benchmark that addresses the limitations of existing evaluation datasets, (2) compare research tools on the same dataset, and (3) compare research and industrial tools.

By studying evaluation data used in the literature, we identify three potential limitations:

- **Age bias:** Firstly, most techniques have not been evaluated on the recent BRs. Previous research relies heavily on the dataset proposed by Lazar et al. [93]. This dataset contains BRs from four projects (`Eclipse`, `Mozilla`, `Netbeans`, and `OpenOffice`) stored in their respective Bugzilla ITSs. All the BRs in these four projects are reported as early as July 1998 and till January 2014. The effectiveness of DBRD techniques on BRs submitted in 1998 should be significantly different from their effectiveness on recent BRs. Clearly, a DBRD technique that works well on data from 10 years ago but no longer so on recent data should not be of much use to developers today. We refer to this potential bias as *age bias*.
- **State bias:** Secondly, as indicated by Xia et al. [169], several fields of a BR may change during its lifetime, e.g., `summary`, `version`, `priority`, etc. Various reasons can lead to changes in BR fields, such as when a bug reporter is new to the open-source project, he might submit a BR where some fields are wrongly assigned. Most studies evaluate the effectiveness of DBRD techniques based on the *latest* states of the fields at the point of data collection. The effectiveness of DBRD techniques should be significantly different if the *initial* states of the fields are used. We refer to this potential bias as *state bias*.
- **ITS bias:** Lastly, as these techniques are commonly evaluated on BRs from one specific ITS (i.e., Bugzilla), it is unclear how they perform on other ITSs (e.g., Jira and GitHub). These ITSs are different in the list of fields that they support, and some DBRD techniques make use of fields that exist in Bugzilla but not others. A DBRD technique specifically designed for a certain ITS data should perform differently when applied in another ITS. We refer to this potential bias as *ITS bias*.

These biases motivate us to investigate our first research question:

RQ1: *How significant are the potential biases on the evaluation of DBRD techniques?*

To answer RQ1, we investigate the performance of the three best-performing solutions identified by Rodrigues et al. [138] in the presence and absence of each potential bias. We demonstrate that the *age bias* and *ITS bias* matter significantly (p-value<0.01) and substantially (large effect size) for all but one case, while the impact of *state bias* is insignificant (p-value>0.05) for all cases.

Based on the above findings, we create a benchmark that addresses age bias and ITS bias and use it to evaluate DBRD techniques that have been shown competitive performance in the research literature. Specifically, we conduct a comparative study that evaluates DBRD techniques on recent BRs (addressing *age bias*) from different ITSs (addressing *ITS bias*). Other than the three techniques mentioned before, we also include two additional recently-proposed DBRD techniques, `DC-CNN` [75], and `HINDBR` [170]. We hereby ask our second research question:

RQ2: *How do state-of-the-art DBRD research tools perform on recent data from diverse ITSs?*

Our result shows that, surprisingly, for most projects, the retrieval-based approach, i.e., `REP` [152], proposed a decade ago, can outperform the recently proposed more advanced and sophisticated models based on deep learning by 22.3% on average in terms of Recall Rate at Top-10 positions (a.k.a. RR@10). This again demonstrates the value of simpler approaches in the saga of simple vs. complex [180, 113, 60].

Further, the research tools have been evaluated in isolation, ignoring tools that have been used in practice. To address this gap, we investigate our third research question:

RQ3: *How do the DBRD approaches proposed in research literature compare to those used in practice?*

To answer RQ3, we compare the five research tools considered in RQ2 with two tools used by practitioners. The first tool is the DBRD technique implemented in the Bugzilla

ITS, named Full-Text Search (FTS). It has been deployed in practice by Mozilla [40]. The next tool is the VSCodeBot, which includes a DBRD feature used in Microsoft's VSCode repository [6]. The experimental results with FTS show that a straightforward duplicate search method used in Mozilla can act as a useful baseline as it can outperform the second best-performing technique on one of the projects in our benchmark by 7.6% in terms of RR@10. Still, the best-performing research tools can boost FTS performance by 22.1% to 62.7%. Moreover, the experimental results on VSCodeBot show that VSCodeBot is better than most tools. Still, the two best-performing research tools can outperform VSCodeBot by 7.6% and 9.8% in terms of RR@5. The results show the value of research on DBRD and the need to develop these research tools further so that practitioners can use and benefit from them.

3.1.1 Background

Several DBRD techniques have been proposed in research to detect duplicate BRs automatically. Yet, the industry uses a different set of tools. However, there is a lack of a systematic study about different approaches for DBRD. Here, we present an overview of BRs in ITSs, DBRD in practice, and DBRD in research.

Bug Reports in Issue Tracking Systems

Existing research [138, 170] heavily uses data from Bugzilla for comparing DBRD techniques. We observe differences across ITSs in submitting BRs, finding duplicates, and marking BRs as duplicates. We select the ITSs in the study based on two factors: whether open-source projects can use them and their popularity. Based on the criteria, we choose Bugzilla, Jira, and GitHub.

Table 3.1: Three example bug reports from `Eclipse` (Bugzilla), `Apache` (Jira), `VSCode` (GitHub) project.

Field	Bugzilla	Jira	GitHub
Bug Id	542516	HIVE-21207	92171
Created	2018-12-07 08:01 EST	04/Feb/19 11:02	7 Mar 2020
Product	Platform	-	-
Component	SWT	None	-
Version	4.8	None	-
Priority	P3	Major	-
Severity	major	-	-
Status	CLOSED	RESOLVED	CLOSED
Resolution	DUPLICATE	Duplicate	-
Summary	Unable to uplaod Image after login	Use 0.12.0 libthrift version in Hive	"C# extension recommended for this file type"
Description	Starting with Eclipse 4.8, horizontal scrolling is ...(*omitted*)	Use 0.12.0 libthrift version in Hive.	Issue Type: Bug Recently, when I open a .cs file. I get the notification... (*omitted*)
Ways to Record Duplicate	Field: <dup_id>530693</dup_id>	Issue Links: is duplicated by Bug HIVE-21173; HIVE-21000	Infered from the comments below: `VSCodeBot` gave a recommendation, the issue reporter acknowledged
URL:	`https://bugs.eclipse.org/bugs/show_bug.cgi?ctype=xml&id=542516`	`https://issues.apache.org/jira/browse/HIVE-21207`	`https://github.com/microsoft/vscode/issues/92171`

'-' : this field is not available in the ITS; 'None' : the value of the corresponding field is empty.

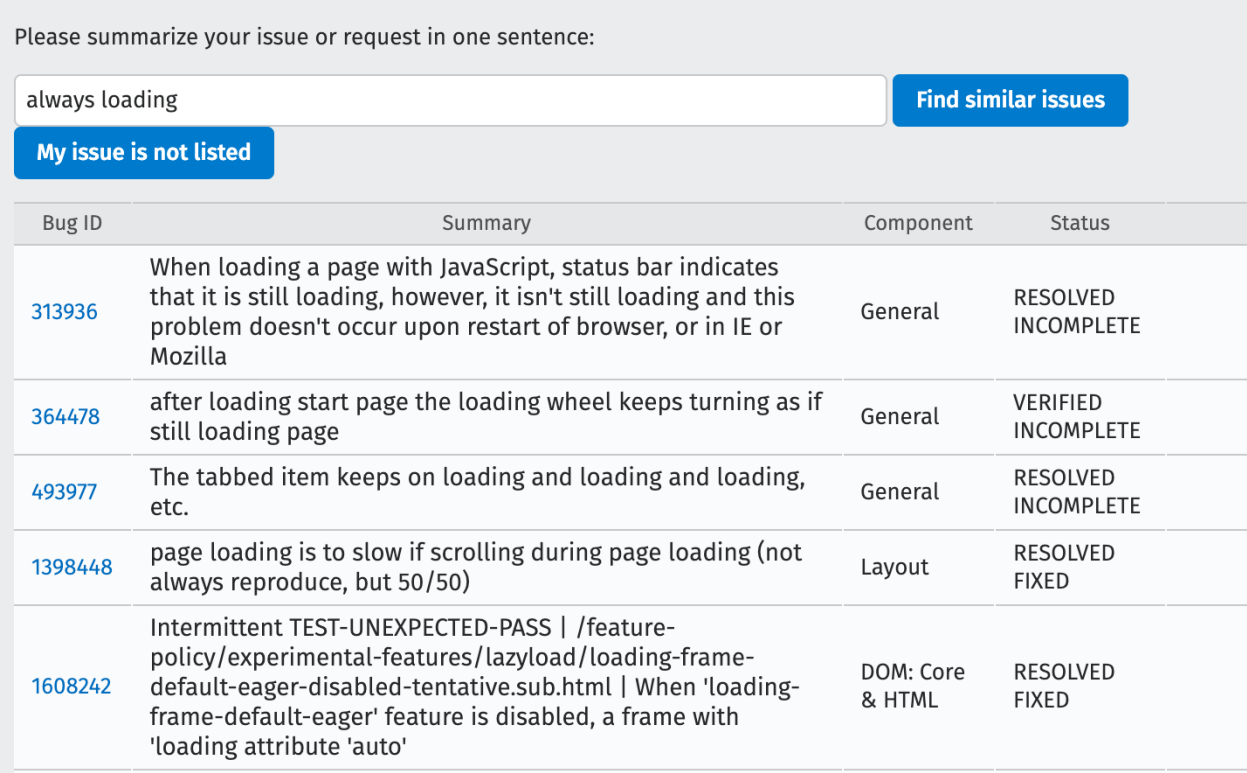
Bugzilla is one of the world-leading free ITSs. Several large projects such as `Mozilla` and `Eclipse` use Bugzilla. A typical bug reporting process involves providing necessary textual and categorical details. A new BR submission in Bugzilla can be seen as a form-filling activity. To mark a BR as duplicate [57] in Bugzilla, users set the `resolution` field to `duplicate` and insert the bug id that this BR duplicates.

Jira has gained popularity over the last decade. According to the Jira official website [3], more than 65,000 organizations used Jira in 2021. Jira follows a form-filling design similar to Bugzilla for creating a new BR. While creating a BR, Jira allows users to relate existing bugs using labels: `is caused by`, `is duplicated by`, or `relates to`. New duplicates are

manually marked using the `resolution` field.

GitHub is a popular Git repository hosting service. It provides rich features, one of which is issue tracking. Issues in GitHub carry a simpler structure when compared to Bugzilla and Jira. Apart from the textual information, the categorical fields are customizable for each repository. Repositories on GitHub can use `label` to categorize issues. However, these labels are not mandatory or well-defined like Bugzilla and Jira. Hence, BRs logged in Bugzilla and Jira have relatively more categorical or structured information. This type of flexibility has pros and cons. On the one hand, it makes it easy to report the issue details; on the other hand, it makes it difficult to extract useful categorical information systematically. Some GitHub repositories provide templates to help users embed categorical information in the textual information. For example, the `VSCode` project provides a textual template that asks users to provide categorical information such as *VSCode version* and *os version*. However, the textual template is not compulsory to follow and can be ignored by issue reporters. The ITS of GitHub uses labels and comments to mark issues as duplicates. We will describe its duplication marking approach in detail in Section 3.1.2.

BRs from different ITSs carry some common fields (such as *BugId*, *Created Date*, etc.) and also some different ones (such as *Severity* that exists in BRs from Bugzilla ITS but does not exist in BRs from Jira and GitHub ITSs). We show one example BR from each of the three ITSs in Table 3.1. As shown in the table, a Bugzilla BR consists of several fields, the types of which can either be textual or categorical. For categorical fields: `Product` usually represents a software product that is shipped, and `Component` is a part of a product. `Severity` states how severe the problem is, while `Priority` is used by the bug assignee to prioritize the issues. `Status` indicates the current state the bug is in, and `Resolution` indicates what happens to this bug. A BR can have several possible `statuses`, such as `unconfirmed`, `confirmed`, `fixed`, `in process`, `resolved`, etc. The possible `resolution` values of a bug can be `duplicate`, `fixed`, `wontfix`, etc. However, Jira does not provide



Bug ID	Summary	Component	Status
313936	When loading a page with JavaScript, status bar indicates that it is still loading, however, it isn't still loading and this problem doesn't occur upon restart of browser, or in IE or Mozilla	General	RESOLVED INCOMPLETE
364478	after loading start page the loading wheel keeps turning as if still loading page	General	VERIFIED INCOMPLETE
493977	The tabbed item keeps on loading and loading and loading, etc.	General	RESOLVED INCOMPLETE
1398448	page loading is to slow if scrolling during page loading (not always reproduce, but 50/50)	Layout	RESOLVED FIXED
1608242	Intermittent TEST-UNEXPECTED-PASS | /feature-policy/experimental-features/lazyload/loading-frame-default-eager-disabled-tentative.sub.html | When 'loading-frame-default-eager' feature is disabled, a frame with 'loading attribute 'auto'	DOM: Core & HTML	RESOLVED FIXED

Figure 3.1: An example of duplicate issue recommendation when typing "always loading" before submitting a new bug report on Mozilla Firefox

exactly the same fields. There are no `Severity` and `Product` fields in Jira. For GitHub, there are no categorical fields. The textual data mainly includes a `summary`, `description`, etc. All the ITSs contain these two common textual fields. The `Summary` is usually a one-sentence text describing a bug. The `Description` usually contains more details, such as the steps to reproduce the bug.

DBRD in Practice

The existence of duplicate BRs causes increasing software maintenance efforts in bug triage and fixing [91]. Several causes can result in duplicate BRs. Prior work [33] shows that duplicate BRs can either be submitted intentionally (e.g., usually when the reporters are frustrated by the same bug not being resolved), or unintentionally (e.g., reporters do not

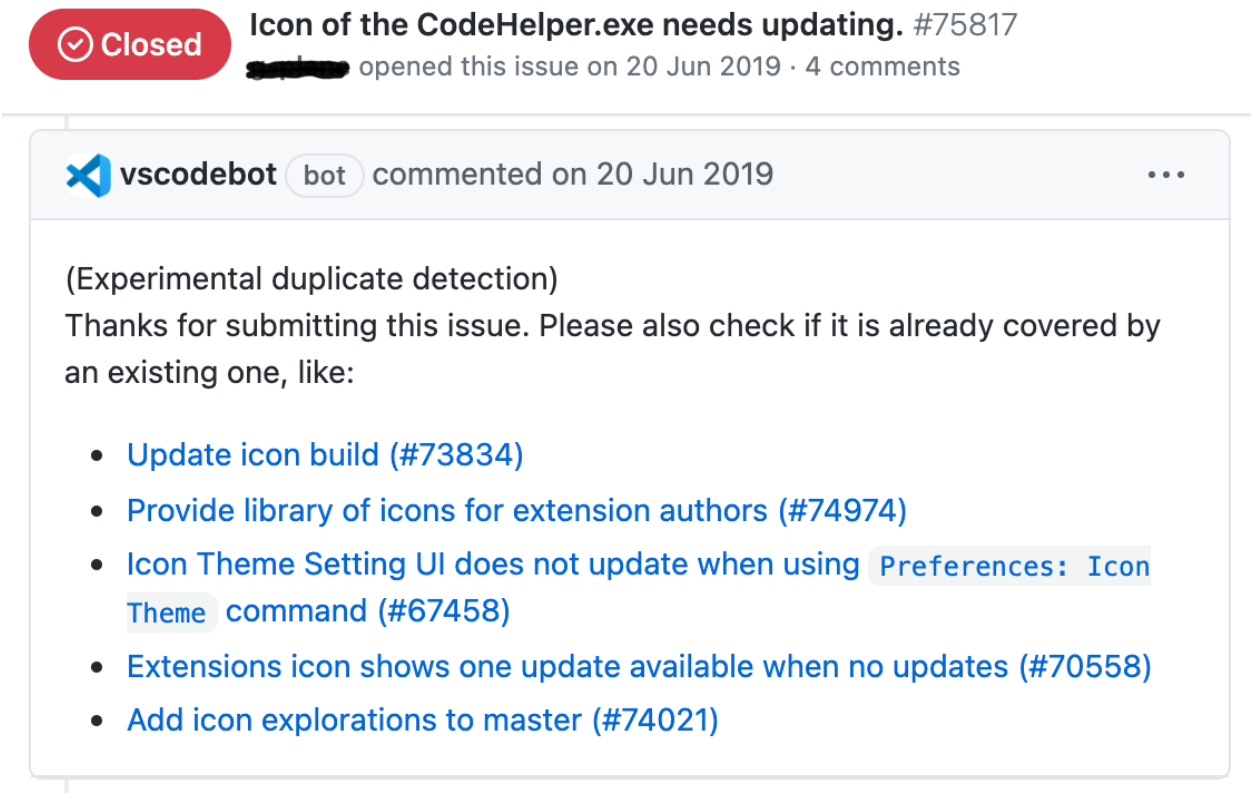


Figure 3.2: An example of `VSCodeBot` duplicate issue recommendation (issue 75817) from Microsoft/VSCode repository

search for existing BRs or cannot identify duplicates due to the lack of experience). To help lighten the workload of bug triagers and avoid redundant bug fixing, DBRD techniques can be useful in two use scenarios. DBRD approaches can either work for (1) *pre-submission* scenario: They can be integrated into ITSs to make bug reporters aware of existing similar BRs and prevent duplicate BRs from happening. Figure 3.1 depicts how the JIT duplicate recommendation works at the pre-submission for `Mozilla` ITSs. or (2) *post-submission* scenario: They can recommend duplicate lists after duplicate BRs are submitted. For example, `VSCodeBot` that is adopted by `VSCode` GitHub repository can recommend a list of potential duplicates. Like other bots [65], `VSCodeBot` also communicates with developers via issue comments and pull request comments. Figure 3.2 shows an example of `VSCodeBot` duplicate issue recommendation on issue 75817[1]. After a user submits the

[1] https://github.com/microsoft/vscode/issues/75817

issue, `VSCodeBot` detects five potential duplicates. The two usage scenarios may require different technical considerations. For the pre-submission usage scenario, the efficiency of DBRD tools plays an important role as it requires DBRD tools to produce a real-time recommendation. Issue reporters are unlikely to be willing to wait for a long time for a DBRD tool to return some results. On the other hand, for the post-submission usage scenario, the DBRD tools can potentially be run overnight in a batch mode to process many BRs, and the emphasis is on optimizing accuracy.

DBRD in Research

Here, we present an overview of the DBRD research tools considered in our work. In the research literature, many DBRD approaches have been proposed and evaluated. In our work, we consider five approaches. They include the three best performers in the study conducted by Rodrigues et al. [138]: `SABD` [138], `REP` [152], and `Siamese Pair` [54]. `REP` is a popular information retrieval-based DBRD approach. `Siamese Pair` is the first deep learning-based DBRD approach proposed in the literature. We further include `HINDBR` [170] and `DC-CNN` [75], which are proposed recently. Since DBRD models mainly differ in how they conduct feature engineering and how they measure similarity between BR pairs, we distinguish these two aspects between different methods in Table 3.2. For detailed explanations of each model, we refer readers to check the original papers. We describe the five approaches in the following paragraphs respectively.

`REP` [152] is a retrieval function to rank BRs based on their similarity with an incoming BR. `REP` considers 7 features, including 2 textual features, i.e., summary and description fields, and 5 categorical features, i.e., product, component, type, priority, and versions. The similarity between the two BRs is a weighted linear combination of the scores of the 7 features. Specifically, `REP` represents texts with uni-grams and bi-grams and calculates the textual similarity by computing an extended version of BM25F [137]. Among the five

Table 3.2: Comparison between different approaches

Approach	Type	Feature Engineering		Distance Measurement
		Embedding	Modeling	
REP [152]	Categorical	-	handcrafted	linear combination
	Textual	-	$BM25F_{ext}$	
Siamese Pair [54]	Categorical	customized	single-layer	Cosine Similarity
	Textual	GloVe	bi-LSTM + CNN	
SABD [138]	Categorical	customized	ReLU	fully-connected layer
	Textual	GloVe	bi-LSTM + attention	fully-connected layer
HINDBR [170]	Categorical	HIN2vec	MLP	Manhattan Distance
	Textual	Word2vec	RNN	
DC-CNN [75]	Categorical	Word2vec	dual-channel CNN	Cosine Similarity
	Textual			

categorical features, if the values of the product, component and type fields of the two BRs are the same, the corresponding feature is 1; otherwise, the feature value is 0. For the remaining two features, i.e., priority and version, the corresponding feature is represented by the inverse of the distance between the values of priority and version of the two BRs. REP has a total of 19 parameters to be tuned, such as the weights of features used. REP proposes to learn these parameters for the bug repositories under consideration using stochastic gradient descent by analyzing a training dataset of historical BRs. The training set of REP is a set of triples (q, rel, irr), where q is the query bug, rel is a duplicate bug with q, and irr is a non-duplicate bug with q.

Siamese Pair [54] is a deep learning architecture combining Siamese LSTM and CNN for DBRD. Siamese Pair first encodes different types of features separately. Specifically, for textual fields, the summary is encoded by a bidirectional LSTM, and a CNN encodes the description. The categorical fields, such as product, priority, and component, are encoded by a feed-forward neural network. In the end, the outputs from the three encoders would be concatenated to represent a BR. The proposed model uses Siamese neural networks and is trained with a max-margin objective. In the evaluation stage, BRs in the test dataset are sorted based on the Cosine similarity with the encoding of the query BR.

Soft Alignment Model for Bug Deduplication (SABD) [138] receives a pair of BRs, a query BR, and a candidate BR. SABD is composed of two sub-networks: one for categorical information and the other for textual information. Each sub-network represents one type of information separately and uses a comparison layer to produce a comparative representation of the two BR vectors either in terms of the values of their categorical or textual fields, respectively. The categorical sub-network is a straightforward dense neural network, while the textual sub-network adopts a sophisticated architecture, where the core is the soft alignment comparison layer. The outputs of the two sub-networks are concatenated. A classifier layer receives the concatenated output and produces the final predicted probability regarding whether the candidate BR is the duplicate of the query BR. The soft-attention alignment exchanges the information between BRs before encoding the textual fields into a fixed-size vector.

HINDBR [170] represents BRs as a Heterogeneous Information Network (HIN). The BRs are connected through their categorical fields. For example, two BRs with bug id x and y that have the same priority, say *Pri_High*, are connected as x – *Pri_High* – y in the HIN. Manhattan distance on the HIN2Vec [59] embedding is used to find semantic similarity in BRs. Xiao et al. introduced three variations of HINDBR: (1) only use unstructured features (i.e., text), (2) only use structured features (i.e., categorical information), and (3) use both. In this study, we found that the models only use unstructured features show better performance than both, thus, we adopted the variant with only text.

Dual-Channel Convolutional Neural Networks (DC-CNN) [75] represents a BR pair with dual-channel matrices. DC-CNN extracts the values of four fields in a BR, i.e., product, component, summary, and description, and treats them as text. After the pre-processing, including tokenization, stemming, and stop word removal, a word2vec [97, 141] model is learned to capture semantic information in the BRs. DC-CNN converts each BR from text to a single-channel matrix based on the word representations learned by the word2vec

model. To represent a BR pair, it combines two single-channel BR representation matrices into dual-channel matrices. The BR pair representations are then fed into a CNN model to capture the correlated semantic relationships between BR pairs. The output of the last layer of the model is used as the predicted similarity score between two BRs.

`REP`, `SABD`, and `Siamese Pair` were evaluated using a *ranking* setting: given a new BR, rank the potential duplicate BRs. `HINDBR` and `DC-CNN` were evaluated using a *classification* setting: given a pair of BRs, decide whether they are duplicates. Rodrigues et al. [138] claim that this classification setting is quite unrealistic since the real scenario presents a much larger set of negative candidates. Furthermore, when a new BR is submitted, all previously submitted reports are duplicate candidates. Thus, they believe the classification-setting highly overestimated performance. Following them and considering how `DBRD` is used in practice (c.f. Section 3.1.1), we evaluate all these approaches using the *ranking* setting.

3.1.2 Data Collection Methodology

Different ITSs represent BRs in different ways, and it is essential to prepare a unified data format to represent the BRs extracted from all of these ITSs. We present our methodology to build the BR dataset from three different ITSs. The methodology involves five main steps: (1) crawling BRs from the website, (2) filtering out open/unresolved BRs, (3) extracting duplicate BR relations, (4) extracting both textual and categorical information, and (5) cleaning and generating duplicate pairs. We elaborate on these steps in the following subsections.

Data Source Selection and Crawling

Project Selection. From each of the three ITSs, we have selected two projects. Here, we describe the projects selected and the rationale behind the selection.

Bugzilla: We choose `Eclipse` [39] and `Mozilla` [40] for Bugzilla, which have the two largest number of issues in the dataset of Lazar et al. [93]. `Eclipse` is an Integrated Development Environment (IDE), and its Bugzilla contains several products, including C Development Tools, Eclipse Modeling Framework, and so on. Similarly, the Mozilla software foundation also includes several projects, including the popular Firefox web browser.

Jira: Following Xie et al. [171], we select two Apache projects hosted on Jira[2], i.e., `Hadoop` [2] and `Spark` [5]. `Hadoop` provides a big data framework for distributed data storage and processing. There are several projects that have been categorized as `Hadoop` in Jira.[3] In this work, we include `Hadoop Common`, `Hadoop HDFS`, `Hadoop MapReduce`, `Hadoop YARN`, `HBase`, `Hive`, and `Hadoop Development Tools` projects. We collectively call them as `Hadoop`. `Spark` is an analytics engine for large-scale data processing.

GitHub: From the datasets listed at GHtorrent [66], we choose the repositories having the largest number of issues. We use the latest update of this `GHtorrent` dataset dated March 6, 2021. Then we manually confirm whether each repository is still in active use. After excluding test or unavailable repositories, we got the five repositories that contained the largest number of issues. They are: `nixos/nixpkgs`, `microsoft/vscode`, `elastic/kibana`, `kubernetes/kubernetes`, and `ansible/ansible`. Among these five

[2]Although Jira is a proprietary bug tracking and project management software, it is free for open-source projects (e.g., the Apache Software Foundation projects) that meet certain criteria [27].

[3]https://issues.apache.org/jira/secure/BrowseProjects.jspa?selectedCategory=10292

repositories, `VSCode` [6] and `Kibana` [4] contained the largest number of duplicate issues in 2018 – 2020, so we included these two projects in our dataset. Visual Studio Code (`VSCode`) is a popular multi-platform source-code editor provided by Microsoft. `Kibana` is a proprietary data visualization dashboard software for Elasticsearch, which is a search engine based on Lucene.

Time Range Selection. We select two different time periods (i.e., old and recent) to investigate whether the age of data impacts the performance of DBRD techniques. For *old* data, we choose January 1, 2012 to December 31, 2014. For *recent* data, we include the recent three-year data from January 1, 2018 to December 31, 2020. The DBRD feature described in Section 3.1.1 was introduced to Bugzilla in 2011. Moreover, the feature adoption date for the projects is unclear. As we would like to focus on analyzing the impact of *age bias* (rather than the impact of DBRD feature introduction), we intentionally picked the time range after this feature was introduced to Bugzilla. Intuitively, if a significant age bias exists considering a 6-year gap period, the bias would have been more pronounced if a longer gap period is considered.

Crawling. For both Bugzilla and Jira, we used the XML export API of the ITSs. For GitHub issues, we used GraphQL API [1] for retrieving the issues in JSON format. Both XML export API and GraphQL API are publicly available. We crawl the issues in June 2021, which is six months later than the aforementioned recent data, to minimize the number of open or unresolved issues.

Filtering Out Open/Unresolved BRs

Following Lazar et al. [93], we only keep *closed* or *resolved* BRs among all the crawled BRs. A typical life cycle of a bug can be abstracted into six main steps [166]: `unconfirmed`,

`new`, `in progress`, `resolved`, `verified`, and `closed`. Note that a *resolved* or *closed* bug can be reopened in the future. Even so, the detailed life cycle in different ITSs may be different. Based on the definition of open bugs by Mozilla [4], we consider a BR as closed if (1) its status is either `resolved` or `verified`; or (2) its resolution is one of the seven types: `fixed`, `invalid`, `wontfix`, `moved`, `duplicate`, `worksforme`, and `incomplete`. For Eclipse, we add another possible status `CLOSED` as closed bugs based on the Eclipse Wiki [5]. For Jira projects, a closed bug has much more possible resolution types than Bugzilla. Thus, we regard a BR as closed if its *status* is marked as either `resolved` or `closed`. For GitHub projects, as it only has *state* field, other than resolution and status. We consider an issue as closed, if its state is `closed`.

Identifying Duplicate BRs

The process to identify whether a BR is a duplicate one is referred to as *duplicate detection*. We use the following strategies to extract ground truths:

Bugzilla: The XML format of each issue in Bugzilla has a field called `dup_id`, which contains a reference to a bug that the current bug is a duplicate of. The dup_id can either be empty or contains another bug's bug_id. If it is not empty, the current bug is a duplicate of the bug in dup_id. Therefore, we directly extract the *dup_id* of those bugs that have a resolution of `DUPLICATE` to gather the duplicate relation.

Jira: Jira does not provide the `dup_id` like Bugzilla does. However, Jira provides rich types of issue dependency links, e.g., `duplicates`, `is duplicated by`, `contains`, to represent the relationship between issues. We identify the duplicate relations by analyzing the `duplicates`, and `is duplicated by` links.

[4] https://wiki.mozilla.org/BMO/UserGuide/BugStatuses

[5] https://wiki.eclipse.org/Bug_Reporting_FAQ#What_is_the_life_cycle_of_a_bug_report.3F

Table 3.3: Textual and categorical fields that are leveraged by the approaches

Fields		REP [152]	Siamese-Pair [54]	SABD [138]	HINDBR [170]	DC-CNN [75]
Textual	summary	✓	✓	✓	✓	✓
	description	✓	✓	✓	✓	✓
Categorical	product	✓	✓	✓	✓	✓
	component	✓	✓	✓	✓	✓
	priority	✓	✓	✓	✓	
	severity		✓	✓	✓	
	type	✓				
	version	✓			✓	

GitHub: Other than the prior two ITSs, the issues in GitHub have a more flexible format. GitHub supports marking duplicates with a comment [64] of the format "Duplicate of #ISSUE_NUMBER". Such comment format can be used to mark the issue that is a duplicate of another issue. As it is not mandatory, we found that not all issue reporters strictly follow this comment format pattern, some users use short-hand notations to mark duplicates, for example, *dup with #ISSUE_NUMBER*. In the end, we use a regular expression that can check the variations of duplicate comments to detect duplicate issues. To validate the reliability of our regular expressions, we manually investigated the duplicate issues detected by the regular expression. We randomly sampled 384 duplicate issue pairs based on the regular expressions from `Kibana` and `VSCode`, respectively. Then, two authors (i.e., investigators) evaluated independently whether the extracted duplicate issues are real duplicates. If there is a disagreement between the investigators, they discuss their investigation results until they reach an agreement. We found only 3 and 21 cases were wrong (i.e., the extracted issues based on the regular expression are not real duplicates) for `Kibana` and `VSCode`, respectively. Thus, our regular expression can detect duplicate issues with at least 94.5% of accuracy. Note we do not check false negatives. As for projects that use Bugzilla and Jira as ITS, false negatives are also possible since people may not mark the duplicates as such.

Information Extraction

All the selected DBRD approaches have taken advantage of both textual and categorical information to improve their effectiveness. Table 3.3 shows the fields that each approach can utilize. The three ITSs have different fields to record categorical information. We extract all the essential fields needed by the approaches, i.e., bug id, categorical fields: `product`, `component`, `severity`, `priority`, `version`, `status`, `resolution`; textual information: `summary`, `description`. We describe how we extract such information from different ITSs.

Bugzilla: A BR on Bugzilla that is exported as an XML file contains clear and well-defined field names. Around 20 fields are given in the XML file, however, not all of them are needed for DBRD. We thus parsed the XML file and saved the essential fields.

Jira: Similar to Bugzilla, Jira has several pre-defined fields for bug reporters to fill in, e.g., `component`, `priority`, `version`. However, Bugzilla and Jira do not have identical categories. For instance, Jira only has a `priority` field to indicate the importance of an issue, while Bugzilla has `severity` and `priority`. Different from Bugzilla, BRs on Jira do not have the `product` and `severity` fields. If any field value is missing, we leave it as an empty string.

GitHub: As mentioned before, GitHub does not provide any well-defined fields for categorical information. Even though labels may provide categorical information, labels are highly customizable for each project. In addition, labels are shared by the other features (e.g., pull requests and discussion) of GitHub[6], so it does not represent categories for issues only. Due to the above limitations, we only extracted textual information from GitHub

[6] https://docs.github.com/en/issues/using-labels-and-milestones-to-track-work/managing-labels

issues. We leave it as future work to investigate how to derive categorical information from GitHub issues.

Data Cleaning

Handling duplicates. It is not rare that one BR has more than one duplicate BRs. For example, the bug 92250 has 45 duplicates[7]. A *group* or *bucket* refers to a set of BRs which are duplicates to each other. The *master* BR is the one to which the rest reports refer. Similar to the prior work [152, 138], we also make the first submitted BR as the *master* and the rest in the bucket as *duplicates*. As we do not cover all the BRs in each ITS and only use the time range of three years, the master BRs of some duplicate BRs can be out of our considered time range (i.e., before January 1, 2012, or January 1, 2018). In that case, we choose the oldest BR in the bucket within the time range as a new master. By this cleaning step, the number of duplicates in our dataset has been smaller than the number of BRs that have been resolved as duplicates.

Basic pre-processing. We conducted data pre-processing in the textual part (summary and description) of a BR: (1) removing punctuations, (2) lower-casing, (3) removing numbers, (4) removing stop words, (5) stemming, and (6) removing single characters. We then tokenized the processed text with white spaces. We reused the script in the replication package provided by Rodrigues et al. [138].

Train-test split. We first sort all the BRs chronologically. Then, we select three years of data. The first two years of data is used for training (including validation, if applicable), while the last year of data is used for testing. Except REP, all the other approaches need validation data. `SABD` and `Siamese Pair` use the last 5% data in training data as validation data, while `HINDBR` and `DC-CNN` use 20% and 10% of the training pairs as validation pairs,

[7] https://bugs.eclipse.org/bugs/duplicates.cgi

Table 3.4: Statistics of data in the six projects

ITS	Project	# BRs	# Dup BRs (%)	# Unique Master BRs	Per bucket	
					Avg. BRs	Max. BRs
Bugzilla	Eclipse	27,583	1,447 (5.2%)	959	2.5	19
	Mozilla	193,587	20,189 (10.4%)	10,702	2.9	151
Jira	Hadoop	14,016	377 (2.7%)	336	2.1	6
	Spark	9,579	354 (3.7%)	290	2.2	14
GitHub	Kibana	17,016	470 (2.8%)	388	2.2	9
	VSCode	62,092	4386 (7%)	2,342	2.9	51

Avg. BRs: average number of BRs in a bucket; Max. BRs: max number of BRs in a bucket

respectively.

One-year time window. For each duplicate BR in the test data, its candidate duplicates are the BRs submitted within one year. We call it *one-year time window* search range. Rodrigues et al. found that despite some minor differences, the findings using a one-year time window are similar to the ones with a longer frame of three years. Thus, we decided to use a one-year time window in our experiments.

Pairs generation. Generating duplicate and non-duplicate pairs is an important implementation detail integrated into each approach. By design, different approaches utilize different ratios of non-duplicate and duplicate pairs. And we follow the original implementation of each approach. Specifically, `REP` generates 30 times more non-duplicate pairs than duplicate pairs. `Siamese Pair` and `SABD` generate the same number of non-duplicate pairs as duplicate pairs. `HINDBR` and `DC-CNN` generate four times more non-duplicate pairs than duplicate pairs. However, the duplicate pairs in the training data are the same, i.e., pair two BRs in the same bucket. We report the number of duplicate pairs in training data in Table 3.4. Note that we would only pair those BRs in the training set.

Overall, we present the basic statistics of the six projects in Table 3.4. We can find that different projects have different characteristics: in one bucket, `Mozilla` can have 151 BRs that are duplicates of each other, while for `Hadoop`, the maximum BRs in one bucket is

only 6. The ranking of the total number of BRs in each ITS is Jira < GitHub < Bugzilla. On average, the percentage of duplicate BRs is also Jira < GitHub < Bugzilla.

3.1.3 Study Setup

We design experiments to answer the RQs described earlier. For RQ1, we focus on understanding the biases that may affect the performance of DBRD techniques. For RQ2, we conduct a comparison among the existing DBRD techniques on our new benchmark. RQ3 focuses on the DBRD techniques in practice.

RQ1: To answer RQ1, we evaluate the three best-performing DBRD techniques reported in a recent study by Rodrigues et al. [138]: `REP` [152], `Siamese Pair` [54], and `SABD` [138]. For the age bias (old vs. recent data) and state bias (initial vs. latest state), we run experiments on the BRs from Bugzilla. For the ITS bias, we run experiments on the BRs from Bugzilla, Jira and GitHub (Bugzilla vs. Jira, and Bugzilla vs. GitHub). To analyze the impact of age bias and state bias, we pick two popular projects (`Mozilla` and `Eclipse`) which use Bugzilla as their ITS as a starting point. We conduct controlled experiments [32, 62] by varying one variable (i.e., age and state) at a time. Furthermore, We choose `Mozilla` and `Eclipse` projects due to the long history associated with them in terms of both the number of BRs and the number of duplicates.

To investigate the impact of *age bias*, we evaluate the effectiveness of the three DBRD techniques on BRs from two time windows: old (2012–2014) vs. recent (2018–2020). Table 3.5 shows the data statistics of these two time windows.

To investigate the impact of *state bias*, we use `Mozilla` and `Eclipse` BRs that were submitted in 2018–2020. For these BRs, we compare the effectiveness of the three DBRD techniques using the latest and initial states of their fields. Among the fields we extracted

Table 3.5: Statistics of old (2012–2014) and recent (2018–2020) data for RQ1

Project	Age	Train		Test	Total	
		# BRs (% Dup)	# Dup Pairs	# BRs (% Dup)	# BRs (% Dup)	# Master BRs
Mozilla	Old	198,653 (9.9%)	35,474	139,502 (9.9%)	338,155 (9.9%)	21,554
	Recent	137,886 (10.1%)	60,498	55,701 (11.2%)	193,587 (10.4%)	10,702
Eclipse	Old	49,355 (5.5%)	4,482	25,021 (12.1%)	74,376 (7.7%)	3,254
	Recent	19,607 (4.7%)	1,725	7,976 (6.5%)	27,583 (5.2%)	959

Table 3.6: The percentage of BRs changed the corresponding state in 2018–2020

Platform	Summary	Description	Product	Component	Priority	Severity	Version
Eclipse	10.8%	-	7.8%	11.7%	1.2%	5.6%	8.6%
Mozilla	11.8%	-	21.4%	24.5%	24.5%	5.4%	4.2%

for experiments, only the description cannot be changed [160], thus we try to recover all the other fields if changed. We recover the initial state of these fields by tracing BR's change history[8]. Each item in the change history of a BR describes the author, time, updated field, removed value, and added value. Table 3.6 shows the percentage of BRs which changed their initial states.

To investigate the impact of *ITS bias*, we evaluate the three DBRD techniques on BRs (reported in 2018–2020) for three sets of projects that use Bugzilla (Eclipse and Mozilla), Jira (Spark and Hadoop) and GitHub (Kibana and VSCode) as ITS.

RQ2: To answer RQ2, other than the three techniques evaluated for answering RQ1, we add two more recent techniques, DC-CNN [75] and HINDBR [170]. These two approaches are initially designed and also evaluated as a classification task. In our experimental setting, all the BRs submitted within a one-year time window are duplicate BRs candidates. For each BR $br_i, i = 1, ..., m$, where m is the total number of BRs in the test set. We pair it with all its candidate BRs, i.e., $(br_i, br_{i,1}), (br_i, br_{i,2})...(br_i, br_{i,k})$, where k is the total number of candidate BRs of br_i, $br_{i,j}, j = 1, ..., k$ is a candidate duplicate of br_i. The trained classification model is used to predict whether each pair is duplicate. The output

[8]An example of change history of a BR: https://bugzilla.mozilla.org/show_activity.cgi?id=122876

Table 3.7: The number of issues per duplicate recommendation frequency by VSCodeBot

# predictions	1	2	3	4	5	Total
# issues	5,016	2,006	1,158	730	2,898	11,808

probability value is then used to rank the possibility that each candidate BR be a duplicate one. If two candidate BRs have the same probability value, we rank them based on their BR ids in ascending order. We get the top-*k* recommendations and evaluate the performance based on the recommendations. We evaluate the five techniques on a new benchmark dataset unaffected by the biases that are found to have a significant and substantial impact on DBRD evaluation in RQ1.

RQ3: To answer RQ3, we investigate the effectiveness of the DBRD techniques used in practice by Mozilla and VSCode projects, and compare them with the aforementioned five DBRD techniques.

Mozilla uses Bugzilla as its ITS, and Bugzilla implements several variants of a DBRD technique named Full-Text Search (FTS)[9]. We study the source code of Bugzilla to identify how FTS works. Simply put, based on the summary input, FTS relies on a BR database and issues SQL queries to search in the database. There are two variants of FTS: FULLTEXT_OR and FULLTEXT_AND. For the first variant, the SQL queries specify 'OR' operations in full-text search; otherwise, the SQL queries specify an exact match. We inferred the variant of the FTS search used by Mozilla (i.e., the OR variant) by trying to enter new BRs into its FTS. We replicated the OR-variant by using the same SQL queries that Bugzilla uses on reading the entire summary field.

Another DBRD technique used in practice is VSCodeBot [7], but its implementation is not publicly available, and it only works for the VSCode repository. Thus, we only evaluate the

[9] https://github.com/bugzilla/bugzilla/blob/5.2/Bugzilla/Bug.pm and https://github.com/bugzilla/bugzilla/blob/5.2/Bugzilla/DB.pm

five techniques on the test data from the VSCode project. Table 3.7 presents the number of VSCode issues that have potential duplicate recommendations by VSCodeBot made in 2018-2020. Among the 62,092 VSCode BRs in our dataset, 11,808 BRs got potential duplicate recommendations. VSCodeBot only recommends up to five recommendations for an issue.

Evaluation Metrics

We evaluate the effectiveness of DBRD tools by calculating Recall Rate@k (RR@k). The evaluation strategy is consistent with the previous works for the DBRD task [54, 142, 165, 153, 25], i.e., only RR@k is used. Note that another metric that has been used in a few DBRD research papers [152, 138] is Mean Average Precision (MAP). MAP concerns the position of the ground truth master BR in each prediction. A higher MAP means that for each BR in the test set, the model can return the ground truth master at a higher place in the ranked result list. The primary reason that prior works, as well as our work, do not consider MAP is that it does not simulate the real usage scenario: In real use, it is unlikely developers would frequently check the recommendations after 10 BRs. For example, prior work on understanding practitioners' expectations on automated fault localization [90] has shown that nearly all the respondents (close to 98%) are unwilling to inspect more than ten program elements to find the faulty code. Another supporting evidence is for VSCodeBot [7] used in the Microsoft Visual Studio Code repository, the largest number of duplicate issue recommendations is 5.

RR@k is defined as follows:

$$RR@k = \frac{n_k}{m},$$

where n_k is the number of duplicate BRs in the test set whose bucket has been found in the top-k positions; m is the total number of duplicate BRs in the test set.

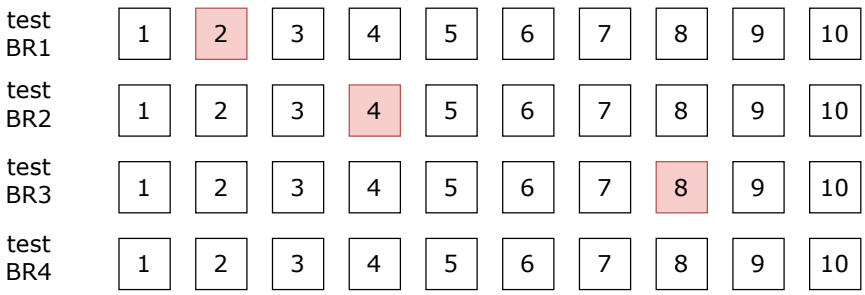


Figure 3.3: Examples of the predictions in the top-10 positions for 4 test BRs.

For example, in Figure 3.3, consider a project with four BRs in the test set. We show the top-10 predictions for the four test BRs. If the corresponding prediction is correct, the box is highlighted in red. In this example, for the test BR1, BR2, BR3, the successful prediction is in the 2nd, 4th, and 8th predictions, respectively. For the test BR4, all the top-10 predictions are wrong. The RR@k in this dataset would be: RR@$k(k = 1) = 0$, RR@$k(k = 2, 3)$=1/4=0.25, RR@$k(k = 4, 5, 6, 7)$=2/4=0.5, and RR@$k(k = 8, 9, 10)$=3/4=0.75.

The process of finding a duplicate BR's master BR is essentially equivalent to finding the bucket to which it belongs. In the prediction stage, for each BR in the test dataset, a DBRD technique queries its candidate BRs and returns a possible duplicate BR list. In the evaluation stage, we group the BRs in the list returned by a DBRD technique into *buckets* list to calculate the recall rate of the predictions. This evaluation strategy is consistent with Rodrigues et al. [138].

In this paragraph, we elaborate on how n_k in RR@k is calculated on the example in Figure 3.3. There are four BRs in the test set. To find the BRs duplicates of a test BR (i.e., the bucket they belong to), instead of only calculating the similarity score of the test BR with master BRs, we compare the test BR with all the BRs in buckets. Like the prior works [152, 138], we also use the highest score among the test BR with all the BRs in the candidate bucket as the similarity score between the BR with the candidate bucket. We illustrate a concrete example as shown in Figure 3.4. We have 100 BRs: { BR-1, BR-2, BR-3, ..., BR-100} and they belong to 20 buckets BR-1 : { BR-1, BR-3, test BR1}, BR-2 : { BR-2,

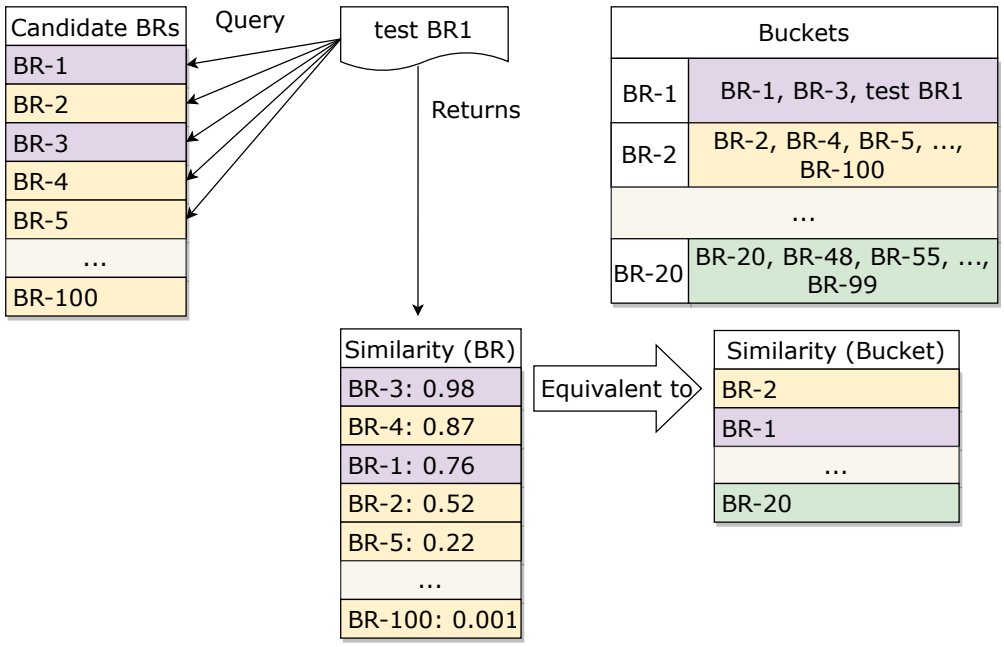


Figure 3.4: The workflow of retrieving the correct bucket.

BR-4, BR-5, ..., BR-100}, BR-3: { BR-3,...}, ... BR-20: {BR-20, BR-48, BR-55, ..., BR-99}. We represent a bucket as a dictionary: the key is the master BR, and the value is its duplicate BRs and itself. Thus, we use the master BR id to refer to the bucket. Given $testBR1$, each model will return a list of BRs sorted by the likelihood of being a duplicate of test BR1. The returned rank list is: [(BR-3, 0.98), (BR-4, 0.87), (BR-1, 0.76), (BR-2, 0.52), (BR-5, 0.22), ... (BR-100, 0.001)]. The first part of each tuple is the BR id, and the second part is the similarity score. According to the ground truth information, test BR1 belongs to the bucket BR-1: { BR-1, BR-3, test BR1}. As test BR1 is in the same bucket as BR-1 and BR-3, the highest similarity scores, i.e., (BR-3, 0.98), will be the similarity score between test BR1 with the bucket BR-1 : { BR-1, BR-3, test BR1}. So, in this example, this model makes a successful prediction at the second position, and we can get $n_k = 1(k = 2)$. With the same strategy, we can calculate the n_k in the rest of the three BRs and get RR@$k(k = 2)$=1/4=0.25.

3.1.4 Study Results

RQ1. How significant are the potential biases on DBRD techniques?

Due to the page constraint, we only present the statistical test results in Table 3.8. The detailed results are available in our online appendix.[10]

Age Bias: We run the Mann-Whitney U test [63] on the following null hypothesis for each pair of DBRD approach and project:

$H_{0.1}$: There is no significant difference in RR@k on old issues and recent issues.

We also compute Cliff's delta (d) effect size [139]. As we have six p-values on the same hypothesis, we also run Bonferroni correction [167], and the significance level α becomes 0.0083. We find that the p-value is < 0.0083 for all approach-project pairs except one case (and thus we can reject the null hypothesis) and the effect size is large. We contend that the age of data significantly affects DBRD performance.

State Bias: Similar to the prior bias, we run the Mann-Whitney U test on the following null hypothesis for each pair of the DBRD approach and project:

$H_{0.2}$: There is no significant difference in RR@k on the initial state and recent state within an issue.

We also compute Cliff's delta effect size. With Bonferroni correction, the significance level α is 0.0083. We find that the p-value is > 0.0083 for all approaches on projects with the initial and latest state (and thus we cannot reject the null hypothesis). We contend that the state does not make a significant difference in DBRD performance.

[10] https://github.com/soarsmu/TOSEM-DBRD

Table 3.8: Mann-Whitney-U with Cliff's Delta Effect Size $|d|$ on RQ1

Bias	Approach	Data	p-value	$\|d\|$
Age	REP	Eclipse	0.003	0.78 (large)
		Mozilla	0.005	0.72 (large)
	Siamese Pair	Eclipse	< 0.001	1 (large)
		Mozilla	0.003	0.76 (large)
	SABD	Eclipse	0.001	0.82 (large)
		Mozilla	0.012	0.66 (large)
State	REP	Eclipse	0.105	0.44 (medium)
		Mozilla	0.190	0.36 (medium)
	Siamese Pair	Eclipse	0.063	0.5 (large)
		Mozilla	0.190	0.36 (medium)
	SABD	Eclipse	0.315	0.28 (small)
		Mozilla	0.315	0.28 (small)
ITS	REP	Jira	0.056	0.36 (medium)
		GitHub	< 0.001	0.66 (large)
	Siamese Pair	Jira	< 0.001	1 (large)
		GitHub	< 0.001	0.97 (large)
	SABD	Jira	< 0.001	0.97 (large)
		GitHub	< 0.001	0.77 (large)

ITS Bias: we also run the Mann-Whitney U test on the following null hypothesis for each approach on Bugzilla vs. Jira, and Bugzilla vs. GitHub data:

$H_{0.3}$: There is no significant difference in RR@k on Bugzilla issues and other ITS issues.

We also compute Cliff's delta effect size. We find that the p-value is < 0.0083 (with Bonferroni correction) for all approaches and the effect size is large (except for one case), and thus we can reject the null hypothesis. Thus, we contend that ITS plays an important role in DBRD technique performance.

Answer 1: Age bias *has a statistically significant impact (with large effect size) on the evaluation of DBRD techniques in all but one cases.* State bias *does not have a statistically significant impact.* ITS bias *has a statistically significant impact (with large effect size) on all but one case.*

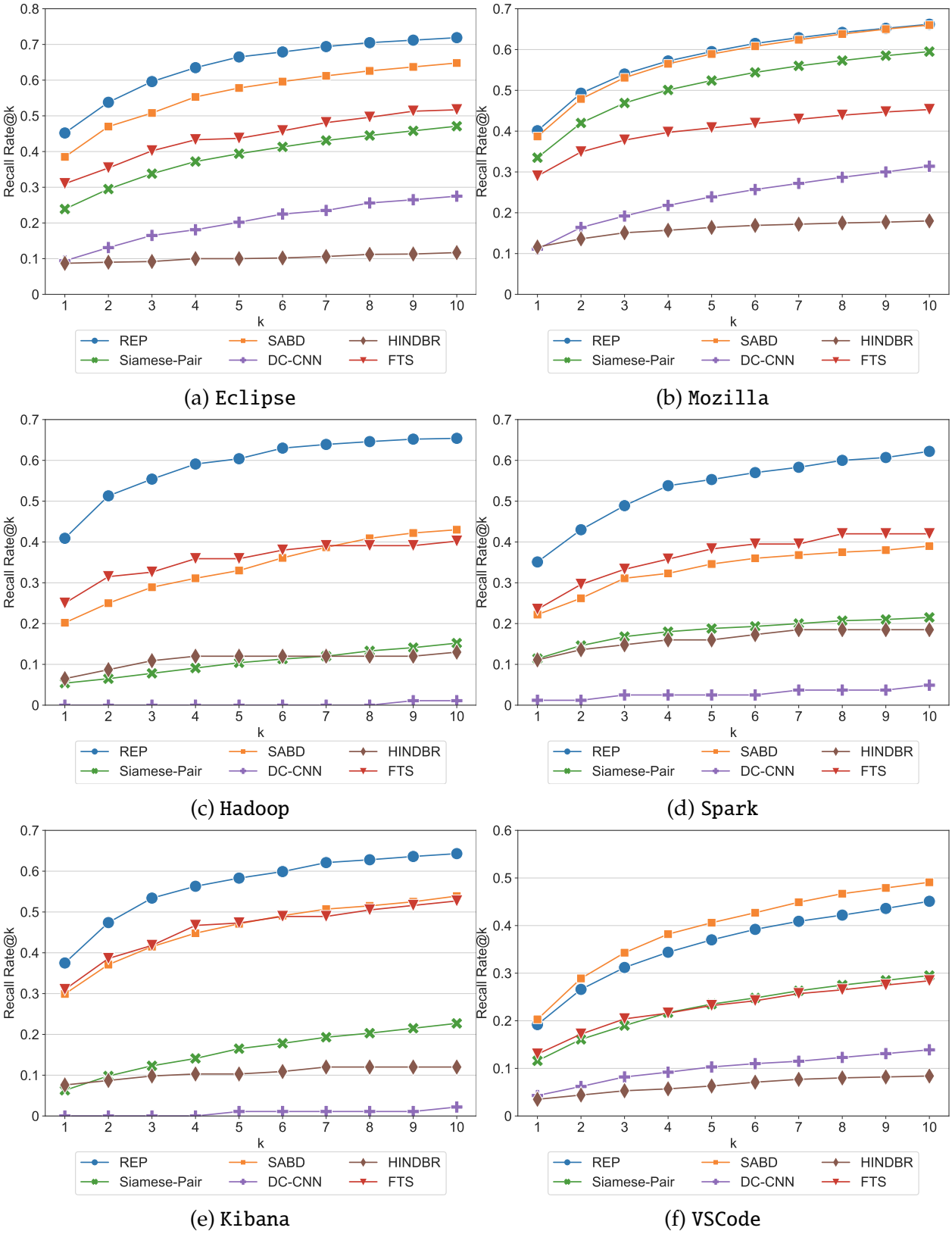
Table 3.9: Statistics of training and testing data

ITS	Project	Train		Test	Total
		# BRs (% Dup)	# Dup Pairs	# BRs (% Dup)	# BRs (% Dup)
Bugzilla	Eclipse	19,607 (4.7%)	1,725	7,976 (6.5%)	27,583 (5.2%)
	Mozilla	137,886 (10.1%)	35,474	55,701 (11.2%)	193,587 (10.4%)
Jira	Hadoop	10,276 (2.8%)	328	3,740 (2.5%)	14,016 (2.7%)
	Spark	6,738 (4%)	414	2,841 (3%)	9,579 (3.7%)
GitHub	Kibana	9,849 (2.9%)	376	7,167 (2.6%)	17,016 (2.8%)
	VSCode	40,801 (7.2%)	9,008	21,291 (6.8%)	62,092 (7%)

RQ2. How do the state-of-the-art DBRD research tools perform on recent data?

Based on the findings from RQ1, we evaluate the existing DBRD techniques on a new benchmark that omits *age bias* and *ITS bias*. Our benchmark contains the recent three-year BRs extracted from Bugzilla, Jira, and GitHub. We build this dataset as described in Section 3.1.2. Table 3.9 shows the statistics of the training and testing data. Figure 3.5 shows RR@k from the 5 approaches on our dataset. The x-axis denotes k values from 1 to 10 while the y-axis shows RR@k. Each approach is highlighted with a different shape and color. For all datasets, REP outperforms the other approaches except for Mozilla and VSCode. On average, REP outperforms SABD by 22.3% in terms of RR@10 across 6 project data. As presented in Table 3.9, both Mozilla and VSCode have the largest number of BRs and duplicates BRs. For these two projects, SABD shows comparable results with REP and even outperforms on VSCode dataset by 9% in terms of RR@10. Siamese Pair, HINDBR, and DC-CNN show fluctuating results. Siamese Pair presents higher RR@k values on Eclipse, Mozilla, and VSCode, while it demonstrates lower RR@k than HINDBR and Hadoop on Spark and Kibana.

Component Analysis. As shown in Figure 3.5, REP is the winner in 5 out of the 6 datasets and it takes advantage of the information from most of the fields. Thus, we investigate REP to understand the contribution of each component in DBRD. REP initializes the weights of the textual features higher than the rest of the features. Additionally, the initial weight of



(a) Eclipse (b) Mozilla

(c) Hadoop (d) Spark

(e) Kibana (f) VSCode

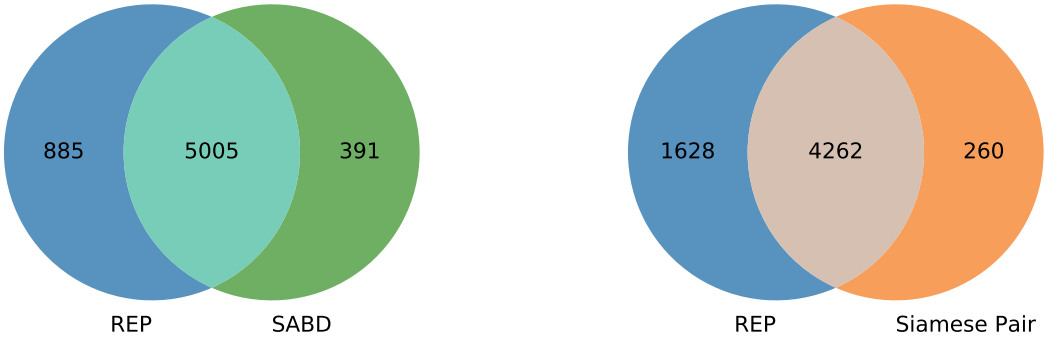
Figure 3.5: Recall Rate@k in the test data of Eclipse, Mozilla, Hadoop, Spark, Kibana, and VSCode

Table 3.10: Investigation of which component benefits REP

RR@k	All	w/o				
		description	short_desc	product	component	priority
1	0.460	0.327	0.350	0.456	0.458	0.450
2	0.544	0.415	0.458	0.527	0.554	0.540
3	0.610	0.456	0.494	0.575	0.598	0.602
4	0.646	0.490	0.510	0.617	0.633	0.637
5	0.673	0.515	0.533	0.644	0.658	0.665
6	0.690	0.531	0.552	0.662	0.663	0.679
7	0.704	0.544	0.569	0.671	0.671	0.694
8	0.706	0.556	0.579	0.681	0.683	0.706
9	0.715	0.565	0.600	0.687	0.683	0.712
10	0.721	0.573	0.613	0.698	0.692	0.717

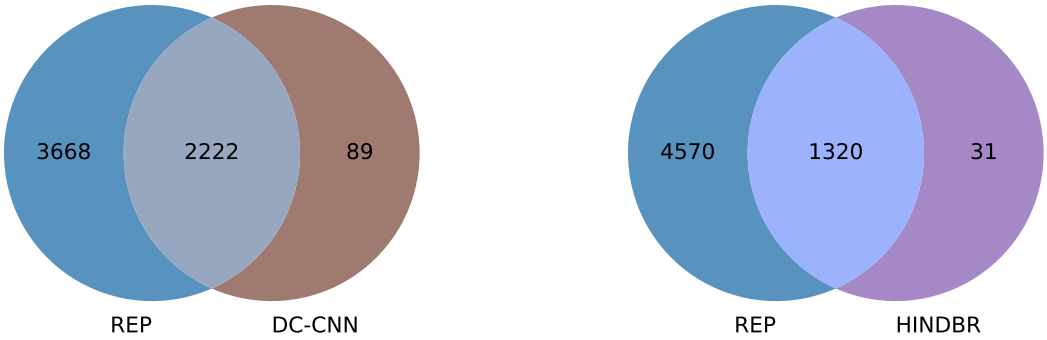
the summary is also higher than the weight of the description. Besides, among the total 19 parameters tuned by gradient descent, 14 parameters relate to the textual fields. Based on the observation, our assumption is that textual fields are the most crucial components among all the fields considered. To further verify our hypothesis, we investigate the contributions of each field value in REP. We run REP on the Eclipse dataset and each time, set a certain field value as empty. Table 10 demonstrates the performances when we set each field as empty. We can find that REP performs the best when all the information is present. When the values of the fields of severity, priority, product, and component are left empty, the performance is similar to the result when all the information is considered. The RR@k is decreased at most 4%. However, when textual information is absent, we can observe that the RR@k decreased at most by 21%. It indicates that textual information plays a more important role than categorical information in DBRD.

Implications. Based on Figure 3.5, when k = 5, the best performing approach can reach RR@k = 0.4-0.6. It indicates that a model can successfully recommend the duplicate BR in the first five positions in 40%-60% of the cases. In other words, in the rest 40%-60% of the cases, the model fails to recommend a duplicate in the first five positions. Among all the projects in our dataset, 2.7% - 10% BRs are duplicates. A technique that has the RR@k



(a) REP vs. SABD

(b) REP vs. Siamese Pair



(c) REP vs. DC-CNN

(d) REP vs. HINDBR

Figure 3.6: REP compared to the other four approaches in terms of successful predictions

of 0.4 - 0.6 means it can help eliminate at most 6% duplicate BRs of all the BRs. Thus, it saves considerable costs and human labor. Besides, considering the large number of candidate BRs, successfully recommending the correct master BR in the first five positions is a challenging task by itself.

Overall, the experiment results show that it is promising to directly adopt certain DBRD approaches designed based on Bugzilla data to other ITS data, such as REP. However, all the approaches demonstrate relatively poor performance in the VSCode dataset. Besides, we find that deep learning-based approaches are less robust than simpler approaches. (1)

REP, which only includes handcrafted features and parameters, is considered as a simpler approach compared to deep learning approaches. REP demonstrated similar performance in both Kibana and Mozilla. However, these two datasets differ in three aspects, i.e., the ITSs, the number of BRs, and the duplicate BR rate: Mozilla contains categorical information while Kibana does not, and Mozilla has 11 times more BRs and nearly 4 times higher duplicate BR rate than Kibana. (2) Since SABD and Siamese Pair show better performances in Bugzilla data than in Jira and GitHub, it implies that it may not be ideal for applying the current deep learning-based DBRD techniques that are specifically designed for one ITS to detect duplicate BRs in different ITSs. (3) Eclipse, Mozilla, and VSCode are the largest projects in terms of the number of BRs and the duplicate BR rate. Deep learning-based SABD can achieve similar or better performance than REP. It infers that deep learning-based approaches favor more training data than simpler approaches. (4) No one can win every battle. Although REP is the best performer in five of the six datasets, it loses to SABD in the VSCode dataset. It suggests that no one design is better than the rest all the time, the performance of DBRD techniques could be subject to the dataset characteristics. Furthermore, we draw the Venn diagrams (Figure 3.6) to demonstrate that each approach can predict duplicates that the best performer cannot.

Answer 2: *REP achieves the comparative results as SABD on the two largest projects and it works well on smaller projects. REP demonstrates promising results, especially on small projects. On the other hand, SABD shows better performance on large projects.*

RQ3. How do the DBRD techniques in research compare to those in practice?

Full-Text Search Line FTS in Figure 3.5 shows the RR@k (for k=[1...10]) across the six projects in comparison with the results for the research tools. We find that FTS is a competitive baseline. It can outperform HINDBR and DC-CNN for all the projects, and it can

also outperform Siamese Pair on four out of six projects. In comparison with SABD (the second-best performing baseline), FTS can achieve similar performance for three out of the six projects. Still, FTS performs worse than REP on all projects. The best-performing research tool (REP) can outperform FTS by 22.1% to 62.7% in terms of RR@10.

VSCodeBot We take the intersection of VSCode BRs shown in Table 3.7 that are truly duplicate BRs and appear in our test set (described in Table 3.9). We then use this data to evaluate the performance of the various DBRD approaches. Figure 3.7 shows the RR@k for each DBRD approach. As VSCodeBot recommends up to five potential duplicate issues, we present RR@k, and the k ranges from 1 to 5. As shown in the figure, the two research tools, i.e., REP, SABD, achieved better performance than VSCodeBot, in terms of RR@5. Meanwhile, Siamese Pair shows a similar result compared to VSCodeBot in terms of RR@5. FTS which is also adopted in practice also shows worse results than VSCodeBot.

Implications. Since FTS is based on exact word matching, the relatively good performance of FTS indicates that many duplicate BRs are more likely to carry the same words in BR titles. It also indicates the important role of textual information. FTS shows poor performance in the VSCode dataset, it shows that the duplicate relationship in VSCode dataset cannot be simply decided based on the words used in BR titles. However, SABD and REP achieve comparable performance as VSCodeBot on the data we investigated, which indicates it is promising to deploy research tools in practice.

Answer 3: *FTS outperforms HINDBR and DC-CNN on all project data, and achieves competitive performance with SABD on three project data. On VSCode BRs, REP and SABD performed better than VSCodeBot. The best-performing research tool (REP) increases the performance of FTS by* 46.1% *in terms of average RR@10, and the performance of VSCodeBot by* 9.8% *in terms of RR@5, respectively.*

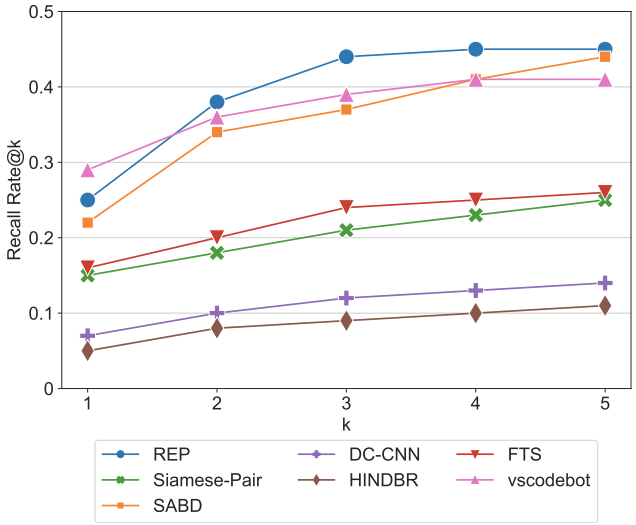


Figure 3.7: Recall Rate@k comparing the tools in research and in practice on the VSCode data

Table 3.11: Age and state bias in an alternative ITS.

Bias	**Data**	**Approach**	p**-value**	$\lvert d \rvert$
Age	Hadoop	REP	0.65	0.13 (negligible)
		Siamese Pair	< 0.001	0.98 (large)
		SABD	0.013	0.67 (large)
State	Hadoop	REP	0.199	0.35 (medium)
		Siamese Pair	0.520	0.18 (small)
		SABD	0.796	0.08 (negligible)
	Spark	REP	0.143	0.4 (medium)
		Siamese Pair	0.058	0.51 (large)
		SABD	0.043	0.54 (large)

3.1.5 Discussion

Based on our empirical results, this section shares our insights to benefit future research on DBRD.

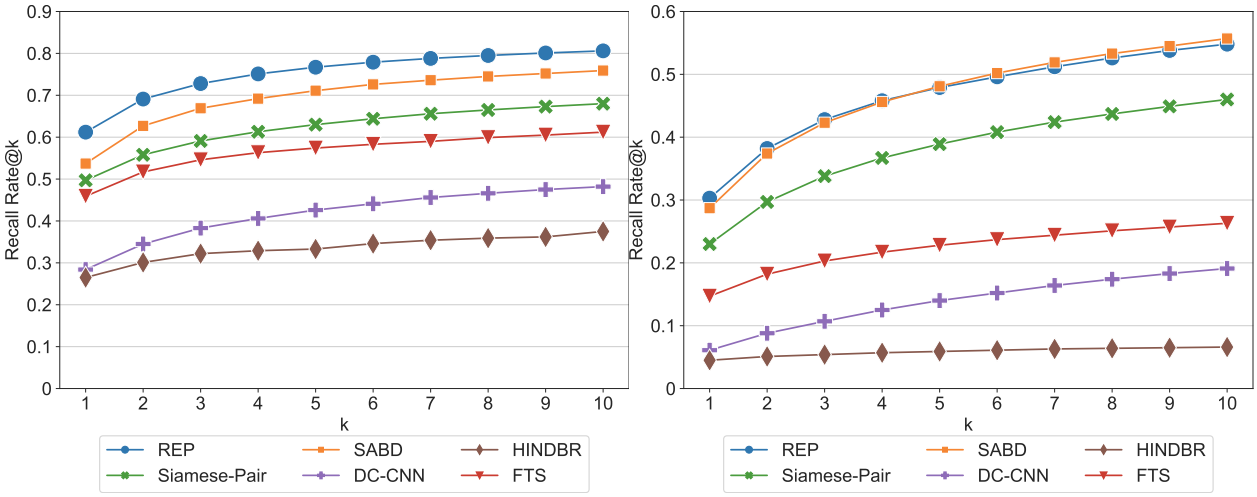
Age/State Bias in an Alternative ITS

As mentioned in Section 3.1.3 Experimental Setup, to investigate the age bias (old vs. recent data) and state bias (initial vs. latest state), we run experiments on the BRs from

Bugzilla. However, it stays unknown whether age bias and state bias exist in another ITS. Here, we investigate the age and state bias on data from an alternative ITS other than Bugzilla.

Age bias in an ITS other than Bugzilla: In RQ1, the experimental results demonstrate that there is a significant difference between tools in research on the old data and recent data from Bugzilla. Besides Bugzilla, we also conducted experiments on the old data from Jira. In RQ1, for Jira, we selected `Hadoop` and `Spark`. However, since the first issue from `Spark` project was created on December 18, 2013. There are not enough old BRs to investigate the age bias. Note that the same reason also applies to all projects selected in our work that use GitHub ITS. Specifically, the first issue from `VScode` project was created on October 14, 2015 and the first issue from `Kibana` project was created on February 6, 2013. Therefore, we are only able to conduct experiments on the old data of `Hadoop`, which uses Jira as an ITS. Following what we did in the RQ1, we evaluate the three tools `REP`, `Siamese Pair`, and `SABD` on the `Hadoop` old dataset (which contains BRs submitted between 2012 and 2014). Table 3.11 shows the statistical test results of the performance of these tools in `Hadoop`'s old and recent data. According to the p-value, we can find that age bias is significant in most cases in `Hadoop`.

State bias in an ITS other than Bugzilla: In RQ1, the experimental results show that there is no significant difference between tools performed with the initial states and the latest states. Besides Bugzilla, we leverage the datasets shared by Montgomery et al. [116] and recover the states of issues in Jira (i.e., `Hadoop` and `Spark`) to the end of the submission day. We then perform the same experiments as RQ1 for state bias and report the results in Table 3.11. As Table 3.11 shows, the state bias is also insignificant in most cases in `Hadoop` and `Spark`, which uses Jira ITS. Note that since the issue change history in GitHub can be deleted, the saved history may be incomplete. Thus, we did not investigate the state bias in GitHub.



(a) Eclipse Old (b) Mozilla Old

Figure 3.8: Recall Rate@k in the test data of Eclipse-Old, Mozilla-Old

Performance Comparison of tools in research and practice on old data.

In RQ3, we demonstrate the performance comparison between tools in research and tools in practice. The experimental results show that `REP` and `SABD` are the best performers. `FTS` is better than `HINDBR` and `DC-CNN`. However, it remains unknown how different DBRD tools compare with each other in the data from old age. For the two tools in practice we evaluated, since `VSCodeBot` is not open-sourced, hence we can only investigate `FTS`. Figure 3.8 shows that `REP` and `SABD` are still the best performers. `Siamese Pair` comes in third. `FTS` is better than `HINDBR` and `DC-CNN` in both datasets. Even so, the performance of all the approaches dropped in `Mozilla`'s old dataset. It suggests that when the size of a dataset becomes larger, the performances of DBRD approaches do not always improve.

Failure Analysis

To understand what are the causes of DBRD approaches that failed to detect some duplicate BRs, we investigated the three best performers, i.e., `REP`, `SABD`, and `Siamese Pair`. We selected the largest projects from each of the ITSs, i.e., `Mozilla`, `Hadoop`, and `VSCode`. We

conducted the following steps to understand the causes of the DBRD failures:

1. Firstly, we get the BRs that were not detected successfully in all five runs (out of 10 positions) by each approach in each dataset.
2. Secondly, we sampled 50 BRs, which failed to recommend the three approaches on the three datasets.

We identified 3 causes of failed duplicate detection and described them as follows.

(1) *Limited or Incomplete description.* When the description is short, it does not provide enough context to understand the issue. Issue reporters attach screenshots or other supporting materials to the issue so that they neglect to write a detailed description. We also found that some descriptions contain too many URLs with only limited textual information. One such example is Bug 1668483 [11] from the `Mozilla` project. The description of this bug is full of long explicit URLs, which makes it hard for models to understand the real content in this issue. Furthermore, we found that issue reporters may break the BR description into multiple parts. They can write a BR description into several comments, which were not considered by our work. One such example is Bug 1641043 [12] from the `Mozilla` dataset. The issue reporter actually described the issue in two consecutive parts, while only the first block is called “description" and the second block is considered as “comment". A complete version of the description may be helpful for DBRD approaches to detect duplicates.

(2) *Inability of the current approaches to understand the different ingredients in BR descriptions.* Since the current DBRD approaches only treat the textual information as unstructured, they cannot extract useful information from the description. The useful information con-

[11] https://bugzilla.mozilla.org/show_bug.cgi?id=1668483

[12] https://bugzilla.mozilla.org/show_bug.cgi?id=1641043

tained in the description may describe the failure, steps to reproduce, system information, etc. They are usually arranged in a structured way. Aside from natural language description, there can also be (1) code snippets, (2) logs, (4) backtraces inside the description. A more reasonable approach may be able to extract different types of information separately. One such example is the duplicate BR pair from the VScode dataset, i.e., issue #105446 [13] and issue #110999 [14]. Both issues contain the system information, steps to reproduce, and screenshots. However, since the information in the descriptions was not considered separately, it may be challenging for models to understand the failures. Besides textual information, a BR can also contain images or screen recordings in the description. However, the approaches evaluated in our work only consider the textual information in BR descriptions. Sometimes, the screenshot shows similar information. For instance, the duplicate issues #108908 [15] and #107104 [16] in VScode repository. Among these two issues, issue #108908 describes the bug as “overlap" and issue #107104 describes the bug as “loads them twice", which does not look duplicate for sure. However, based on the screenshots from both issues, we can find these two issues refer to the same bug. An approach that can handle the screenshots and screen recordings would be helpful in these cases.

(3) *Different failures with the same underlying fault.* As indicated by Runeson et al. [142], there are two types of duplicates: (1) they describe the same failure; (2) they describe two different failures with the same underlying fault. We also encountered difficult cases when both BR described the issue correctly, however, they described the two different failures while the underlying fault is the same. Since the current approaches are based on the similarity of BRs, it is challenging for them to detect the second type of duplicates.

[13] https://github.com/microsoft/vscode/issues/105446

[14] https://github.com/microsoft/vscode/issues/110999

[15] https://github.com/microsoft/vscode/issues/108908

[16] https://github.com/microsoft/vscode/issues/107104

One such example is the duplicate BR pair from the Hadoop dataset, i.e., HBASE-24609 [17] and HBASE-24608 [18]. The two BRs describe two different objects, i.e., `MetaTableAccessor` and `CatalogAccessor`. Even for developers with some experience on Hadoop projects, it may not be possible to recognize that they are duplicates.

Lessons Learned

Age bias and ITS bias should be considered for DBRD, and even for other tasks that involve BRs. We show that two kinds of bias (age and ITS) affect the performance of DBRD techniques. These biases must be considered while designing and evaluating future DBRD techniques. Further, we believe that any task involving BRs, e.g., bug localization [130, 109, 89], bug severity prediction [26, 105, 28], and bug triage [174, 151] etc. should also provide due consideration for these biases. When evaluating an approach, it would be better to consider the diversity of the ITS.

Using `FTS` and `REP` as a baseline for evaluating DBRD approaches. We observe that `FTS` although simple, outperforms many other DBRD approaches for most projects (all except `Mozilla`). `REP`, although proposed a decade ago, is the overall best performer. Thus, we suggest future research include these simpler techniques as baselines. Future state-of-the-art approaches need to demonstrate superior performance over these simpler techniques.

Choose your weapon - Projects with a medium to low volume of historical BRs may not benefit from deep learning-based tools. The two best-performing tools are `REP` and `SABD`. `SABD` is deep learning-based, while `REP` is not. Comparing the performance

[17] https://issues.apache.org/jira/browse/HBASE-24609

[18] https://issues.apache.org/jira/browse/HBASE-24608

Table 3.12: Mann-Whitney-U with Cliff's Delta Effect Size $|d|$ on RQ1 with controlling the size of training/validation pairs

Bias	Data	# Sampled pairs Training	# Sampled pairs Validation	Approach	p-value	$\|d\|$
Age	Eclipse	3,342	108	REP	0.002	0.78 (large)
				Siamese Pair	< 0.001	0.9 (large)
				SABD	0.040	0.54 (large)
	Mozilla	68,396	2,418	REP	0.003	0.76 (large)
				Siamese Pair	0.002	0.8 (large)
				SABD	0.007	0.7 (large)
ITS	Jira	626	26	REP	0.047	0.37 (medium)
				Siamese Pair	< 0.001	0.81 (large)
				SABD	< 0.001	0.79 (large)
	GitHub	724	28	REP	0.029	0.41 (medium)
				Siamese Pair	< 0.001	0.93 (large)
				SABD	0.010	0.4775 (large)

of both tools in Figure 3.5, we can notice that their performance is similar for projects with the largest number of BRs (Mozilla and vscode). However, there is a clear big gap in performance for the other projects (although they still contain thousands of BRs as training data). This suggests that the applicability of deep learning-based solutions may be limited to very large ITSs with tens of thousands of BRs submitted over a few year period (considering *age bias* and data drift phenomenon [143]). For most ITSs, non-deep learning-based approaches may outperform. Note that, in our experiments, we did not use all the historical data for training since our findings in RQ1 show that there is a significant difference when applying a DBRD approach to old data and recent data. Besides, the old and recent data carry different characteristics, e.g., the number of BRs, so the predictions of the models trained in the past data may become less accurate in the recent data [194]. In addition, when training with more data, the training process takes longer and is more computationally expensive.

As shown in Table 3.9, we identified that the number of BRs in different projects varies a lot (i.e., the number of training and validation pairs are different). The size of training

data might be a confounding factor. To understand whether our findings still hold when we have the same number of training and validation pairs, we investigated the impact of the data size. We adopted the pair generation strategy used by SABD [138]. The positive pairs are all combinations of the BRs which belong to the same bucket. On the other hand, the negative pairs are randomly generated by pairing a BR from one bucket with the BR from another bucket. Since the number of positive pairs is fixed, we generated the same number of negative pairs. For each bias and each dataset, we sampled the same number of training pairs and validation pairs. For instance, when we work on age bias on the Eclipse dataset, as the old dataset has 8,668 BR pairs, while the recent dataset contains 3,342 BR pairs, we would sample the same number of 3,342 BR pairs from the old dataset (i.e., downsampling to the minority label). The number of training and validation pairs are reported in Table 3.12. We then conducted the same experiment for RQ1 with the sampled pairs as presented in Table 3.12. The p-values regard each bias are all cases. The detailed results can be found in our replication package. [19] In the table, we find that the main message of this paper is still valid even if we work on the same number of training and validation pairs. Please note that the experimental data for state bias have no difference in terms of size (i.e., the numbers of BRs for before and after state are the same), so we excluded it in this additional analysis.

Future research approaches should compare with industry tools. Researchers have largely ignored the comparison of DBRD techniques with industry tools. We conducted experiments on both `FTS` and `vscodebot`. Our experiments showed that `FTS` and `VSCodeBot` can outperform many research tools. While we have highlighted the need for evaluation with industry tools in the context of DBRD, we believe our suggestion is valid even for other SE tasks, too. Researchers should investigate if some alternative tools are used in practice to solve the same/similar pain points and compare the performance of research

[19] https://github.com/soarsmu/TOSEM-DBRD

Table 3.13: Averaged seconds of per prediction used by different approaches in the test data of Eclipse, Mozilla, Hadoop, Spark, Kibana, and VSCode

ITS	Project	REP	Siamese Pair	SABD	DC-CNN	HINDBR	FTS
Bugzilla	Eclipse	0.07	0.15	1.61	2.49	0.76	0.20
	Mozilla	0.25	0.89	12.63	16.64	5.80	1.74
Jira	Hadoop	0.07	0.07	0.80	1.19	0.30	0.10
	Spark	0.13	0.05	0.69	1.02	0.28	0.08
GitHub	Kibana	0.07	0.18	1.40	2.10	0.67	0.19
	VSCode	0.08	0.70	3.53	5.56	1.92	0.40

tools with those "defacto" tools.

Efficiency matters for the pre-submission DBRD scenario. In the post-submission scenario, the DBRD technique has the liberty of time to predict duplicates, but it is not the case for the pre-submission scenario. The DBRD response time varies depending on the number of BRs in the ITS. JIT duplicate recommendation used by Bugzilla, i.e., FTS, works faster than most research tools as they only query the summary field of existing BRs. In usability engineering, a response time of over 1 second is considered to interrupt a user's flow of thought [120]. Given that users can perceive a delay difference of 100 milliseconds [38, 120], some DBRD approaches, which take over 10 seconds to predict potential duplicates, do not seem to meet the requirements. We report the seconds per prediction spent by each approach. The experiments were run on a machine with Intel(R) Xeon(R) Gold 5218R CPU @ 2.10GHz (Mem: 252G) with 4 GeForce RTX 2080Ti (11G). Only one GPU was utilized when running a single deep learning model. From Table 3.13, we can find that the time needed for each approach to make a prediction differs in different projects. Generally speaking, for tools in research, REP, and Siamese Pair are faster than the rest approaches, while in the largest project Mozilla, the run time difference is more pronounced. REP is 3.56× faster than Siamese Pair, 50.52× faster than SABD, and 66.56× faster than DC-CNN. For the tools in practice, since VSCodeBot is not open-sourced, we cannot measure its run time on the pre-submission scenario. For FTS, we can find that it

is faster than most research tools.

We suggest two potential topics for future DBRD research: (1) investigating the acceptable delay for the pre-submission DBRD scenario and (2) optimizing DBRD response time. To reduce the prediction time, future research can also consider reducing the search space. For example, instead of including all the BRs submitted within the one-year time window as candidates, further approaches can reduce the candidates first: applying a time-efficient technique, such as BM25, to filter out the BRs with a low chance of being duplicated. After that, expensive deep learning-based models can be used.

Comments should be considered. Based on our failure analysis, we found that comments in the BRs may be helpful. Especially, when the issue reporter separates the description of a BR into several parts. In the post-submission scenario, leveraging comments can provide additional information for DBRD tools to represent a BR.

Different ingredients in the description should be handled separately. Although in Bugzilla and Jira, there are dedicated fields for categorical information, we found that the description can be arranged in a structured way. It can have steps to reproduce, expected behaviour, and observed behaviour etc. Issue reporters usually include images or videos inside the description. An approach that can understand different contents from the description would be beneficial. For GitHub, where the information such as system information, extensions used, and steps to reproduce are usually included in the description, an approach that can extract all the useful information from the description would be more effective.

Other resources in the project can be considered to improve DBRD accuracy further. The current DBRD approaches are designed for detecting BRs with similar contents. If future approaches want to tackle the duplicates that have different failures, we suggest they consider other resources in the project, such as code base, to understand the relationship

between different failures with the same root cause.

3.2 New Approach

Over the past decades, various DBRD approaches have been proposed [152, 54, 138, 75, 170]. With the rapid development of deep learning, many deep learning-based approaches have been proposed in recent years [138, 75, 170]. They have demonstrated superior performance when the bug repositories are large enough to train the deep learning models. For instance, SABD [138] achieved over 0.6 in Recall Rate@20 in all the experimented datasets. One of the common characteristics of these datasets is that they all contain more than 80k bug reports and over 10k *duplicate* bug reports in the training data, which is large enough to train a deep learning model. It is well acknowledged that deep learning models require a large amount of data to achieve high precision [136]. However, bug repositories of many projects are not large enough to train a deep learning model.

Based on the dataset provided by Joshi et al. [87], it was discovered that out of the 994 studied GitHub projects that have more than 50 stars and forks, the average number of issues was 2,365. Additionally, it is interesting to note that many active projects, including those with more than 100k stars, have fewer than 10k issues. For example, till 5th May 2023, both `ohmyzsh/ohmyzsh` [17] and `axios/axios` [10] have around 4k issues each, while `vuejs/vue` [21] has around 10k issues. Therefore, we argue that most projects do not have tens of thousands of issues. The repositories with tens of thousands of issues are considered as *atypical*, while a *typical* repository contains less than or around 10k issues. It is essential to highlight that young and fast-growing projects, although currently having a few issues, require more attention in handling the DBRD challenge. For instance, `Significant-Gravitas/Auto-GPT` [18], which was initially released on March 30, 2023, now contains less than 2k issues while it gets 124k stars. How to improve the performance

of DBRD in the *typical* bug repositories remains an open problem.

Before the development of deep learning, many non-deep learning-based approaches have been proposed [142, 83, 152, 153] (we refer to them as "*traditional* approaches" in this dissertation). These approaches are more promising for detecting duplicate BRs in typical bug repositories than deep learning-based approaches. However, traditional approaches rely on either the vector space model [142] or the bag-of-words model [83]. These models cannot capture the semantics of BRs. We seek to improve the performance of non-deep learning-based approaches by considering the semantics of BRs.

Recently, bLLMs, e.g., Vicuna [49], LLaMA 2 [159], and ChatGPT [15], have achieved outstanding performance in a multitude of NLP tasks [55, 37, 131]. However, leveraging the potential of bLLMs to improve DBRD's performance is not trivial. The most straightforward way is to directly query bLLMs on whether two BRs are duplicates. However, this is impractical due to the following reasons.

(1) Time-consuming and costly. To obtain the potential master BRs to which a given BR may be duplicated, we must pair it with all the BRs available in the repository. All previously submitted BRs are considered duplicate candidates when a new BR is submitted. It is infeasible to query bLLMs to compare the given BR with all the BRs in the repository, as the bLLMs' response is not instantaneous. While speeding it up is possible (e.g., by running many queries at once), it quickly gets very costly for bLLMs such as ChatGPT, which operates on a pay-per-use basis for their API usage.

(2) Ignorance of other BRs in the repository. If a method only compares two BRs at a time, it will not take into account the information present in the other BRs stored in the repository. Therefore, it would be hard to decide the relative order of all the duplicate candidates in order to recommend the top-k duplicate candidates. Although one possibility is in addition to querying ChatGPT on whether the BR pair is duplicated or not, we ask

ChatGPT to provide a measure of how confident it is in its answer, expressed as a similarity score or confidence score. However, without considering the information from other BRs, the similarity score will be less reliable.

(3) bLLMs are generative AI techniques which are designed to generate contents. Although bLLMs have achieved impressive performance in a multitude of NLP tasks, many researchers argue that bLLMs are only good at language abilities but not at actual reasoning [111, 31]. Thus, to take full advantage of bLLMs, we carefully design the task to ensure its suitability for bLLMs. As DBRD requires some reasoning on how two BRs are duplicated to each other, it is not suitable to query bLLMs directly.

In this study, we present Cupid, which stands for leveraging ChatGPT for more accurate duplicate bug report detection. Cupid aims to tackle the abovementioned challenges when directly querying LLMs for DBRD. We propose to leverage LLMs as an intermediate step to improve the performance of the traditional DBRD approach. Based on our earlier benchmarking study, REP [152] demonstrates the best performance in the datasets with a typical number of issues, which is also the focus of this work. Thus, we select REP as the backbone duplicate retrieval method. Specifically, Cupid leverages state-of-the-art ChatGPT to identify keywords from bug reports and then incorporate them with REP to achieve better performance. By doing so, Cupid avoids using ChatGPT to compare the given bug report with all the bug reports in the repository. Furthermore, by standing on the shoulder of the traditional DBRD approach, Cupid also considers the information of the other bug reports in the repository. In particular, $BM25F_{ext}$ used by REP calculates inverse document frequency (IDF), which is a global term-weighting scheme across all the bug reports. In addition, Cupid prompts ChatGPT to identify keywords from bug reports, which requests ChatGPT to generate a list of relevant keywords based on the content of a bug report. Keyword identification is closer to a generative task than a decision-making task.

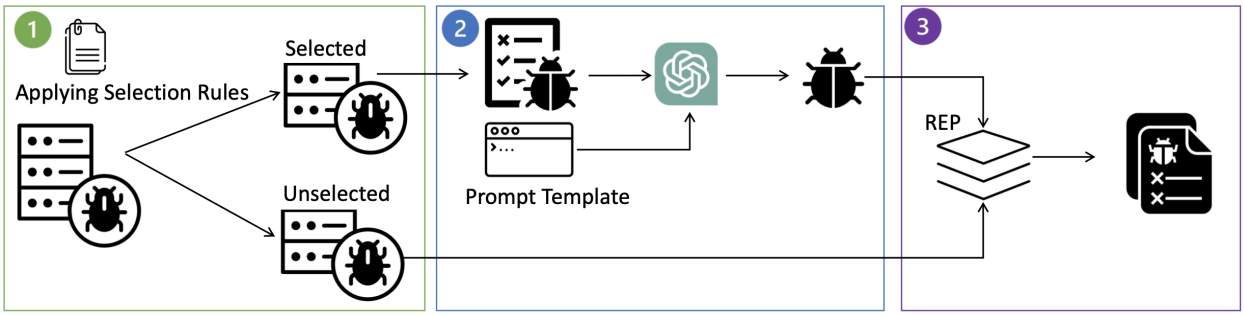


Figure 3.9: Cupid contains three stages: In Stage 1, it applies selection rules to select the test BRs that need to be processed; In Stage 2, it utilizes ChatGPT to process the selected BRs; In Stage 3, it leverages REP to retrieve potential master BR for each test BR.

3.2.1 Cupid

We propose Cupid to combine the advantages of both the traditional DBRD approach and LLM. As mentioned earlier, our work focuses on solving the DBRD challenge in the repositories with a typical number of issues; evidence shows that traditional DBRD approaches would fit more in this condition than deep learning-based approaches [183]. Figure 3.9 shows an overview of the proposed method. The overall process consists of three main stages: (1) Applying *selection rules* to select the BRs that need to be processed by ChatGPT, (2) Running ChatGPT with *prompt template* to get the essential keywords of the selected BRs, and (3) Applying REP to retrieve *potential master BRs*.

In the following sections, we first introduce the datasets used in this work. Then, we describe the selection rules and prompt template used by Cupid. Finally, we introduce the REP approach.

Applying Selection Rules

Considering the computational cost of ChatGPT, we did not run ChatGPT on all the BRs in the test dataset. Similarly, in practice, we do not need to run ChatGPT on each newly submitted BR. We explore and propose selection rules to improve efficiency while keeping

accuracy. These rules are based on the length and content of the BRs, to prioritize BRs that are harder to process by REP while reducing the number of BRs fed into ChatGPT. The selection criteria are as follows:

Length: We select BRs whose description is considered to be *long*. We consider BRs whose description is longer than n words as *long* BRs. We get n by calculating the 75th percentile of the description length in the training set. We select long BRs because long BRs are usually not concise and contain long stack traces and code snippets. These long BRs would make it challenging for REP to retrieve the potential master BRs.

Content: We select the BRs whose description contains code snippets or URLs. We use regular expressions to match and select these BRs. The choice of regular expressions is due to their simplicity and effectiveness in matching specific patterns like URLs and code snippets within text. We leave applying advanced techniques such as island grammars for future work.

Note that after keeping long BRs, we still have BRs that contain code snippets or URLs. Some BRs are very short and with the majority of the content being code snippets or URLs. For developers, this information is useful. However, for a DBRD method, this information can be hard to process. We select these BRs because not all the code snippets and URLs are useful for REP to retrieve the potential master BRs. We also do not directly remove code snippets or URLs. The reason is that we want to keep the original structure of the BRs for ChatGPT to understand the language better. We then utilize ChatGPT to identify keywords from these BRs.

Prompt Template:

```
I have a BR which contains summary and description. I want you to select
    keywords from both parts which keep the main meaning of the BR. These
    keywords would be used for duplicate BR detection. Output format: 'Summary
    : Selected Keywords \n Description: Selected Keywords' \n\n >>> Summary:
    [Summary] \n\n >>> Description:
[Description]
```

Running ChatGPT with Prompt Template

After Stage 1, we run ChatGPT on the selected BRs, i.e., either (1) the `description` is long or (2) the `description` contains code snippets or URLs.

Prompt [104] is a set of instructions that can be used to probe bLLMs to generate the target outcome [31]. Prior studies have empirically shown that ChatGPT is sensitive to prompts. Thus, the prompts should be carefully designed for different tasks to enable bLLMs to demonstrate their abilities.

We craft the prompt template used by Cupid as shown below.

This template is designed for a single-turn dialogue. For each BR, we open a new dialogue with ChatGPT. After getting the response from ChatGPT, we replace the original `Summary` and `Description` in the BR with the returned identified keywords of `Summary` and `Description`. We keep the remaining part of the BR unchanged.

Regarding the design of the prompt template, our intuition is that we consider BRers are likely to have more expertise and domain knowledge than ChatGPT. Therefore, the language and terms they use when reporting bugs may be similar to each other, and DBRD methods can leverage this similarity. It would benefit more not to replace the whole expression but rather select and keep the essential information for DBRD methods to process. We also conduct experiments with other prompt templates to support our intuition and report the results in Section 3.1.4.

Retrieving Potential Master Bug Reports

Considering the superiority of REP in the task of DBRD shown in a recent study [183], especially on projects with a typical number of issues, we use REP as the DBRD approach in Cupid. Here, we briefly introduce the REP approach to make the dissertation self-contained. We refer the readers to the original paper [152] for more details.

As shown in Formula 3.1, REP is a linear combination of seven features, including **textual** features and **categorical** features.

$$REP(d,q) = \sum_{i=1}^{7} w_i \cdot feature_i \tag{3.1}$$

, where d is the BR in the repository R, q, is the query (i.e., new BR), w_i is the weight of the i-th feature, and $feature_i$ is the i-th feature. The first two features are both **textual** features, and the rest five features are **categorical** features. Figure 3.2 shows how to get

each feature.

$$
\begin{aligned}
feature_1(d,q) &= BM25F_{ext}(d,q) \text{ //of unigrams} \\
feature_2(d,q) &= BM25F_{ext}(d,q) \text{ //of bigrams} \\
feature_3(d,q) &= \begin{cases} 1, & \text{if } d \cdot prod = q \cdot prod \\ 0, & \text{otherwise} \end{cases} \\
feature_4(d,q) &= \begin{cases} 1, & \text{if } d.comp = q.comp \\ 0, & \text{otherwise} \end{cases} \\
feature_5(d,q) &= \begin{cases} 1, & \text{if } d.type = q.type \\ 0, & \text{otherwise} \end{cases} \\
feature_6(d,q) &= \frac{1}{1+\mid d.prio - q.prio \mid} \\
feature_7(d,q) &= \frac{1}{1+\mid d.vers - q.vers \mid}
\end{aligned}
\tag{3.2}
$$

The first two features regard the textual similarity between two BRs over the fields `summary` and `description`. These two textual features are calculated by $BM25F_{ext}$ between BR d and query BR q. $BM25F$ [137, 179] is an effective textual similarity function for retrieving documents that have structures. The authors of REP extend $BM25F$ by considering term frequencies in queries and proposed $BM25F_{ext}$.

In $feature_1$, `summary` and `description` are represented in uni-gram, while in $feature_2$, `summary` and `description` are represented in bi-gram. Thus, the input of $BM25F_{ext}$ consists of a bag of uni-grams and bi-grams in both features. For $feature_{3-5}$, they are the categorical features of `product`, `component`, and `type`, respectively. If the corresponding field value from d and q is the same, the value of the feature is 1, otherwise, it is 0. For $feature_{6-7}$, they are the categorical features of `priority` and `version`, respectively. They

are calculated by the reciprocal of the distance between the corresponding field value from d and q. Overall, the REP approach contains 19 free parameters with different initial values. These parameters are tuned by gradient descent.

3.2.2 Study Setup

Research Questions

To understand whether Cupid performs better compared to existing state-of-the-art approaches and whether each component of Cupid is useful, we answer the following two research questions (RQs):

- **RQ1:** *How effective is* Cupid *compared to the state-of-the-art approaches?*
- **RQ2:** *How effective are the components of* Cupid*?* To answer this RQ, we conduct an ablation study on the components of Cupid. This RQ is further divided into the following sub-RQs:
  - **RQ2.1:** *How effective is the prompt template?*
  - **RQ2.2:** *How effective are the selection rules?*
  - **RQ2.3:** *How effective is ChatGPT compared to other bLLMs?*

Dataset

As mentioned in Section 3.2, we are concerned about boosting the performance of DBRD, especially in the bug repositories with the typical number of issues. Therefore, the target datasets contain a typical number of issues. We employ three datasets, i.e., Spark, Hadoop, and Kibana datasets, which are provided by a recent benchmarking study by Zhang et

Table 3.14: Dataset statistics. Cupid here refers to the selected BRs run by ChatGPT.

Dataset	Total Bugs	Train. Pairs	Valid. Pairs	Test	
				Dup. Bugs	Cupid
Spark	9,579	626	26	81	59
Hadoop	14,016	626	27	92	57
Kibana	17,016	724	28	184	114

al. [183]. These datasets contain around 10k issues each, which is considered a typical number of issues. These datasets are recent issues, ranging from 2018 to 2022, which addressed the age bias, i.e., the model performs differently on the recent data and old data. Spark and Hadoop are two popular open-source distributed computing frameworks. They both use Jira as their ITS. Kibana is a visualization tool for Elasticsearch, and it uses GitHub as its ITS. The statistics of the datasets are shown in Table 3.14. The duplicate and non-duplicate pairs were sampled by Zhang et al. [183]. Their ratio is 1:1. We obtained the data in the dataset provided by Zhang et al. In our experiment, we fixed the number of training and validation pairs. The number of duplicate BRs in the test set is the BRs we investigate. We report the performance of each approach in terms of how they perform in retrieving the master BRs.

Evaluation Metrics

We adopt the same evaluation metrics as indicated in Section 3.1.3.

Compared Techniques

In this work, we compare Cupid with state-of-the-art DBRD techniques, which consider DBRD as a ranking problem, i.e., REP [152], Siamese Pair [54], and SABD [138]. Section 3.1.1 contains a brief introduction to these methods.

In RQ2.3, we also compare ChatGPT with other open-source bLLMs. We select three bLLMs, i.e., Vicuna, WizardLM, and Llama 2-Chat based on their performance in the MMLU benchmark on the chatbot leaderboard [20] in August 2023. Due to the limit of computing resources, we could only run the bLLMs containing less or equal to 13B parameters. We introduce these bLLMs in Section 2.2.1.

ChatGPT Setup

Given that ChatGPT is still fast evolving, it has undergone several iterations [12]. In this study, we worked on the GPT-3.5 version. To interact with ChatGPT, we used an open-sourced API [9] that creates a chat window on the ChatGPT website. It saved us from the manual labor of opening a chat window and copying the response back. Although an official ChatGPT API is available, we cannot use it without paying for it. Therefore, we chose to use the free version of ChatGPT, which we believe has a wider range of users than the paid one. As such, our results would be more valuable as they apply to a wider range of users.

During the experiments, for each query BR, we initialize a new conversation to avoid the influence of the previous conversation on other BRs. Since ChatGPT may generate different answers for the same query, we ran ChatGPT five times for each query and aggregated the results (i.e., summing up the 5-round results) to obtain the final answer.

Implementation

To fairly compare Cupid with the baselines, we fix the training pairs for all techniques. Since there is randomness in the deep learning-based models, i.e., Siamese Pair and

[20] https://huggingface.co/spaces/lmsys/chatbot-arena-leaderboard

Table 3.15: RR@k obtained on the Spark dataset. The best performance in terms of RR@10 is highlighted accordingly.

Method	RR@1	RR@2	RR@3	RR@4	RR@5	RR@10
REP	0.346	0.383	0.457	0.481	0.481	0.556
Siamese Pair	0.037	0.049	0.059	0.064	0.074	0.121
SABD	0.202	0.247	0.281	0.294	0.304	0.331
Cupid	0.346	0.395	0.432	0.469	0.481	0.593

Table 3.16: RR@k obtained on the Hadoop dataset. The best performance in terms of RR@10 is highlighted accordingly.

Method	RR@1	RR@2	RR@3	RR@4	RR@5	RR@10
REP	0.402	0.489	0.522	0.554	0.576	0.609
Siamese Pair	0.033	0.046	0.057	0.063	0.076	0.093
SABD	0.215	0.267	0.293	0.304	0.324	0.411
Cupid	0.391	0.511	0.565	0.576	0.609	0.652

SABD, the reported results were the average results after running them five times. The implementation details can be found in our replication package.[21]

3.2.3 Study Results

RQ1: Comparing with baselines

Table 3.15, 3.16, and 3.17 show the results of Cupid and the baselines on the Spark, Hadoop, and Kibana datasets. Overall, Cupid consistently improves the DBRD performance in terms of RR@10 on all three datasets, yielding an improvement of 6.7% (Spark) to 8.7% (Kibana) over the prior state-of-the-art approach REP. This improvement is obtained by successfully utilizing the language generation ability of ChatGPT to transform the BRs into a format where only essential information is kept. In comparison with the best-performing

[21] https://github.com/soarsmu/Cupid

Table 3.17: RR@k obtained on the Kibana dataset. The best performance in terms of RR@10 is highlighted accordingly.

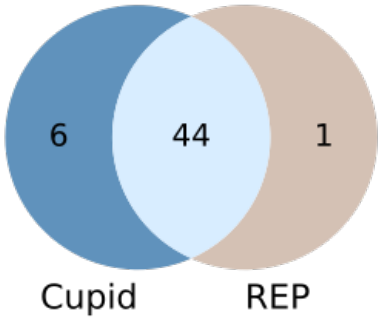
Method	RR@1	RR@2	RR@3	RR@4	RR@5	RR@10
REP	0.364	0.440	0.527	0.560	0.587	0.620
Siamese Pair	0.020	0.036	0.050	0.063	0.076	0.092
SABD	0.293	0.382	0.428	0.467	0.489	0.555
Cupid	0.408	0.522	0.571	0.603	0.62	0.674

deep learning-based approach, i.e., SABD, we observe an improvement of up to 79.2% on the Spark dataset. In the low-volume datasets, SABD and Siamese Pair lose to non-deep learning approaches, i.e., REP and Cupid.

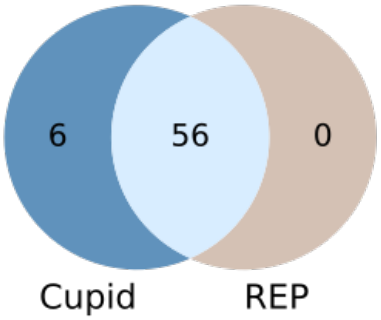
Comparing the performance of Siamese Pair and SABD in all three datasets, we can find that Siamese Pair suffers more from the challenge of limited training data. Siamese Pair performs less than 50% of SABD in all the three datasets in terms of RR@10. We argue that when there is a lack of adequate training data, comparing different deep learning-based models is less meaningful.

Dataset-wise, all approaches perform relatively worse on the Spark dataset and relatively better on the Kibana dataset. The observation aligns with the findings from prior studies [138]: the same DBRD approach, i.e., SABD, achieves a variety of RR@10 on different datasets examined, ranging from 0.55 (on OpenOffice dataset) to 0.7 (on Netbeans dataset). It shows that the performance of a DBRD technique also depends on the dataset characteristics. This observation inspires us that it would be beneficial for each dataset if we tune the prompt template based on the characteristics of each dataset. We leave this for future work to boost the performance further.

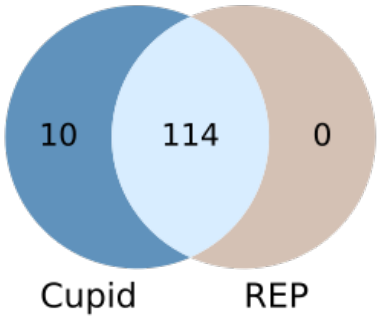
Figure 3.10 shows the Venn diagrams for successful predictions made by the prior state-of-the-art method, i.e., REP, and Cupid on each dataset and all datasets combined. We see that Cupid successfully retrieves more master BRs compared to REP. On the Hadoop



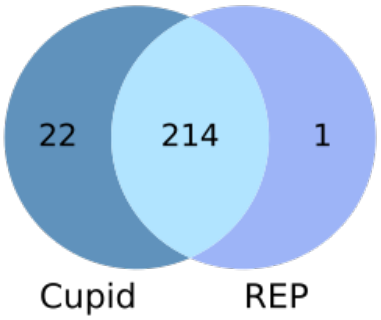
(a) on Spark dataset



(b) on Hadoop dataset



(c) on Kibana dataset



(d) All the three datasets

Figure 3.10: Successful prediction Venn diagram

and Kibana datasets, only Cupid successfully retrieved more master BRs, while REP did not successfully retrieve more.

To demonstrate the ability of Cupid, we show an example, i.e., the query BR is `HADOOP-17091`[14] where REP failed to predict the correct master BR in the top-10 positions, while Cupid managed to. Figure 3.11 shows the `summary` and `description` of this issue. We can see that there is no natural language in the description, containing only error messages. Thus, REP considered the most possible master BR to be `HADOOP-16648`, which also contains a large portion of the error messages. We checked the single-run result by ChatGPT. Thanks to the language understanding and generation ability of ChatGPT, Cupid identified the keywords: `Javadoc, HTML version, HTML4, HTML5, warning, comments, valid, GeneratedMessageV3, package, not found, error` from the `description` of `HADOOP-17091`. The generated shorter `description` on the query BR has several words overlap with the

Figure 3.11: The case where Cupid succeeded while REP failed: HADOOP-17091

`description` of the real master BR (`HADOOP-16862`). It enables Cupid to rank at the first position successfully. Since the real master BR has a long error message as the `description`, REP failed to retrieve it. This example shows that ChatGPT can be helpful when descriptions are long and contain non-natural language texts. It can generate the most important keywords, which are vital for duplicate detection.

Based on Figure 3.6, we can also observe that on the Spark dataset, REP can actually predict a BR, i.e., `SPARK-33661` [20] that Cupid fails to predict. Figure 3.12 shows the `summary` and `description` part of this issue. We checked the single-run result by ChatGPT. The identified keywords are: `Summary: Unable, load, RandomForestClassificationModel, trained, Spark 2.x, Description: load, RandomForestClassificationModel, trained, Spark 2.x, Spark 3.x, exception, raised, schema incompatibility, saved, data, expected, existing, random forest models, upgrade, retrain`. The selected keywords look reasonable since they keep the important identities that are related to this bug. The mas-

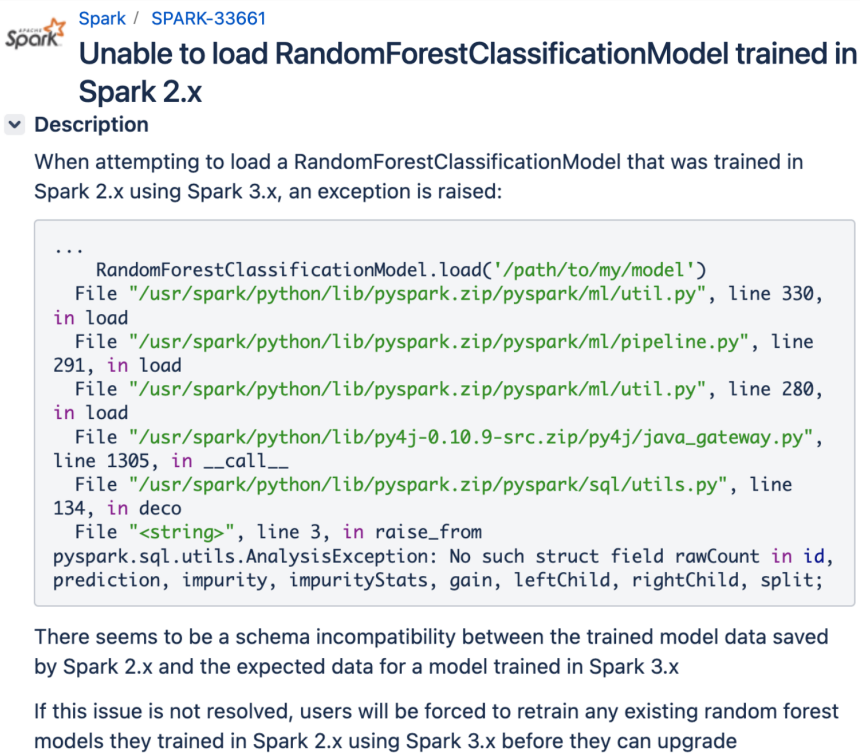


Figure 3.12: The case where Cupid failed while REP succeeded: SPARK-33661

ter BR, which was listed at the top-1 position by Cupid is `SPARK-31169` [19]. This BR is about different results that are obtained when building random forest models using different versions of Spark. It is clear that it is not a duplicate of `SPARK-33661`. However, it is not hard to find that in the `summary` of this issue, common words exist, such as `Random Forest`, `SparkML 2.3.3 vs 2.4.x`. In addition, in the `description` part, we can find similar words, e.g., `train,version,spark,model`, which are quite relevant to the identified keywords in `SPARK-33661`. This example shows that, in some cases, keywords are not sufficient to identify duplicate BRs. The same set of words may lead to different errors.

Answer to RQ1: Cupid outperforms the best baseline by 6.7%, 7%, and 8.7% in terms of RR@10 on the Spark, Hadoop, and Kibana datasets, respectively.

Listing 3.1: Prompt Template 1

Prompt Template 1:

```
Rephrase the following BR in five distinct styles, to avoid repetition, while
    keeping its meaning. Output format: '[1-5]. Summary: [Rephrased Summary]
  \n Description: [Rephrased Description]' \n\n >>> Summary: [SUMMARY] \n\n
    >>> Description: [DESCRIPTION]
```

RQ2: Ablation Study

RQ2.1: The effectiveness of Prompt Templates. We first investigate the effectiveness of different prompt templates on the Spark dataset due to the fact that it is the smallest dataset. We came up with a basic prompt template (i.e., Prompt Template 1), as shown in Listing 3.1.

In **Prompt Template 1**, we aim to describe the task and requirement in a simple way. We also specify the output format. The hypothesis is different stakeholders, e.g., users, developers, or testers, have different expertise and experience levels; thus, how they write BRs would vary. Therefore, we prompt ChatGPT to get alternative BRs. This procedure can be viewed as data augmentation [51], where the goal is to generate auxiliary samples that are semantically similar to the original sample. In the beginning, we believed prompting ChatGPT to `rephrase` the BR should be one of the most direct ways to achieve this goal.

Thus, we further experimented with a more comprehensive **Prompt Template 2**, where we added a persona description and also included the aim of this step. This prompt template is an augmented version of **Prompt Template 1**. Listing 3.2 shows the template.

Table 3.18 shows the results of querying ChatGPT with the two templates above and with the template employed in Cupid. We observe that **Prompt Template 2**, which is more comprehensive than **Prompt Template 1**, indeed leads to a slightly better performance: it surpassed the method with Prompt Template 1 by 2.3% in terms of RR@10. Although these two templates convey very similar meanings, with one being more succinct and

Listing 3.2: Prompt Template 2

Prompt Template 2:

```
I want you to act as a collaborator for maintaining BRs from a software
    project. Your job is to rephrase the BR, to avoid repetition, while
    keeping its meaning. The aim of this step is to help filter duplicate BRs.
     You will need to write five different versions of the BR you encounter.
    You can delete the contents you perceive useless. Output format: '[1-5].
    Summary: [Rephrased Summary] \n Description: [Rephrased Description]'. Now
    , your need to rephrase the following BR: \n\n >>> Summary: [SUMMARY] \n\
    n >>> Description: [DESCRIPTION]
```

the other being more verbose, they did make an impact on the performance of running rephrased BRs in DBRD. Using the final template in Cupid can boost the performance of these two prompt templates by 11.7% in terms of RR@10. These results indicate the significance of prompts.

At first glance, both Prompt Templates 1 and 2 seem more intuitive than the final prompt template we use (the one shown in Section 3.2.1). Prompt Template 1 and 2 queried ChatGPT to *rephrase*, while the final prompt in Cupid queried ChatGPT to *select keywords*. However, after checking the rephrased BRs generated by ChatGPT with Prompt Template 1 and 2, we believe *rephrasing* the test BRs is not the right direction to pursue. In retrospect, it would make more sense to rephrase all BRs in the dataset, regardless of training or testing. However, as mentioned in Section 3.2, the drawback is the expenses of running ChatGPT. There is a widely-experienced error: `Too many requests in 1 hour, try again later` [8], which many users complain about. Despite the lack of an official document specifying the exact number of requests that can be made with ChatGPT per hour, this issue commonly occurs, hindering the whole DBRD process. Given the major difference between the query BR and candidate BRs, only rephrasing query BRs would not make it easier to retrieve the master BRs. In the context of traditional DBRD approaches, it could make the distance between the rephrased BR and the master BR further.

RQ2.2: The effectiveness of Selection Rules. Here, we also conducted experiments on

Table 3.18: Ablation study on different prompt templates: RR@k obtained on the Spark dataset. The best performance in terms of RR@10 is highlighted accordingly. PT is short for "prompt template".

Method	RR@1	RR@2	RR@3	RR@4	RR@5	RR@10
w/ PT 1	0.309	0.37	0.432	0.457	0.469	0.519
w/ PT 2	0.333	0.37	0.432	0.457	0.469	0.531
Cupid	0.346	0.395	0.432	0.469	0.481	0.593

Table 3.19: Number of test bugs and bugs that need to run after adopting selection rules.

Selection Rule	Dataset		
	Spark	Hadoop	Kibana
None	81	92	184
Length	33	25	77
Length→Content	59	57	114

Spark dataset to investigate the effectiveness of selection rules.

Table 3.19 shows how many BRs need to query ChatGPT after adopting (1) no-selection rules, (2) selection by length, and (3) selection first by length and then content. If we only use *length* as the selection criteria, we will only need to run ChatGPT on 40.7%, 27.2%, and 41.8% of the original test BRs in Spark, Hadoop, and Kibana dataset, respectively. While the computational cost would be reduced, it is essentially a trade-off: we want to achieve both efficiency and accuracy, which can be contradictory in some cases. We want to take full advantage of ChatGPT with minimal computational costs. Other than length, we also identify the *content* criteria. After adopting both length and content criteria, the BRs needed to be processed by ChatGPT increased and accounted for 72.8%, 62%, 62%, which still saved more than 25% BRs from processing.

Table 3.20 shows the corresponding results of applying selection rules. Comparing the performance of no selection rules, i.e., querying all the test BRs with ChatGPT, and applying *both* selection rules, we can observe that after applying the rules, RR@10 improves by 2.2%. Despite only making a small improvement, it frees at least 25% of the BRs in the

Table 3.20: Ablation study on selection rules: RR@k obtained on the Spark dataset.

Selection	RR@1	RR@2	RR@3	RR@4	RR@5	RR@10
None	0.333	0.395	0.432	0.469	0.481	0.58
Length	0.333	0.37	0.395	0.444	0.457	0.556
Length→Content	0.346	0.395	0.432	0.469	0.481	0.593

test set from querying ChatGPT. Here, we do not only save the computational cost but also improve accuracy.

It is surprising to find that by only applying the length criteria, the performance actually drops. Table VI shows that in the Spark dataset, only 40.7% are considered long. Thus, more than half of bug reports remain the same without querying ChatGPT. However, other than the lengthy bug reports, short bug reports with little useful information are also confusing for REP. It is not rare that bug reporters paste error messages or code snippets or even directly a URL with little meaningful text as the description. ChatGPT works like a filter in these short bug reports to exclude useless information for the DBRD task. It thus makes sense that if we increase the number of bug reports processed by ChatGPT from 33 to 59, the performance increases by 6.7% in terms of RR@10. This result again shows the importance of trade-offs in efficiency and accuracy. Ultimately, if we include both filtering rules, the resulting method can achieve the best performance. It showcases the effectiveness of the filtering rules.

RQ2.3: The effectiveness of ChatGPT. Here, we have conducted experiments on the Spark dataset to assess the efficacy of ChatGPT in comparison to other open-source bLLMs. In a similar manner, we executed all three bLLMs five times, while the generated responses across all five runs remained the same. We adapted the final version of the prompt template from Cupid, making slight modifications to ensure compatibility with the appropriate prompt format for each LLM. Listing 3.3 shows the prompt templates employed by each LLM. The main element, denoted as *select keywords*, remains consistent across all templates,

Listing 3.3: Prompt Template for Vicuna, WizardLM and Llama 2-Chat

Prompt Template for Vicuna and WizardLM:

```
"A chat between a curious user and an artificial intelligence assistant. The
    assistant gives helpful, detailed, and polite answers to the user's
    questions.\n\nUSER: I will given you a BR summary and a BR description.
    You need extract the usefull keywords from summary and description,
    respectively. These keywords would be used for duplicate BR detection. You
     need to reply like this:\nSummary Keywords:\nDescription Keywords:\
    nASSISTANT: Sure!</s>\nUSER:Bug report summary: {}\n\nBug report
    description: {}\n\nASSISTANT: Summary Keywords:",
```

Prompt Template for Llama 2-Chat:

```
"<<SYS>>\nA chat between a curious user and an artificial intelligence
    assistant. The assistant gives helpful, detailed, and polite answers to
    the user's questions.\n<</SYS>>\n[INST]\nUser: I will given you a BR
    summary and a BR description. Note that the description can include log/
    error message or stack traces. You need extract the usefull keywords from
    summary and description, respectively. These keywords would be used for
    duplicate BR detection. You need to reply like this:\nSummary Keywords:\
    nDescription Keywords:\nBug report summary: {}\n\nBug report description:
    {}\n\n[/INST]\n"
```

Table 3.21: Ablation study on ChatGPT: RR@k obtained on the Spark dataset.

Model	RR@1	RR@2	RR@3	RR@4	RR@5	RR@10
Vicuna	0.358	0.395	0.432	0.432	0.457	0.506
WizardLM	0.370	0.395	0.420	0.444	0.469	0.568
Llama 2-Chat	0.296	0.358	0.383	0.407	0.42	0.494
ChatGPT	0.333	0.395	0.432	0.469	0.481	0.58

while only the formats differ.

Table 3.21 shows the results with the comparison among ChatGPT and the other three bLLMs. We can observe that WizardLM can achieve a similar performance as ChatGPT with only 2% drop in terms of RR@10. Vicuna and Llama 2-Chat perform worse compared to ChatGPT. The good performance of WizardLM makes it promising to use an open-source LLM for the DBRD task. Further investigation on when and why open-source bLLMs lose to ChatGPT can be put to take full advantage of the latest advancement of bLLMs.

Answer to RQ2: Both prompt templates and selection rules are effective in improving the performance of Cupid. Furthermore, ChatGPT is better than the three selected open-source bLLMs.

3.2.4 Threats to Validity

Internal. The main internal threat is whether there is data leakage in ChatGPT. However, since we do not have access to ChatGPT's training data, we cannot verify whether there is data leakage in ChatGPT. Even so, since we did not directly use ChatGPT to compare whether two BRs are duplicates, instead, we utilize ChatGPT indirectly, which may not benefit much from memorizing the training data. Furthermore, we noticed that ChatGPT did not exhibit unrealistic perfect performance, e.g., reaching 0.9 at RR@10, across different prompt structures. It suggests that it is less likely for ChatGPT to rely solely on the memorization of its training data. Thus, we believe this threat is minimal.

External. The primary external threat is the generalizability of our findings. This study focuses on datasets with a typical number of BRs, roughly $10k$ issues. Therefore, our results may not extend to datasets with a significantly greater number of BRs, such as those containing tens of thousands of issues. Nevertheless, we believe that our findings remain valuable for the majority of projects. This is supported by the fact that in a dataset of 994 high-quality projects from GitHub, each project contains an average of $2k$ issues [87]. Another external threat is that ChatGPT is fast-evolving. Our results may not be generalized to future ChatGPT versions.

3.3 Summary

In the benchmark work, we evaluated DBRD techniques both in research and practice. We analyzed the factors affecting DBRD performance. We showed that on recent data, DBRD approaches demonstrated significantly different performance when compared to their performance on old data. Clearly, a DBRD technique that works well on data from many years ago but no longer so on recent data, should not be useful to developers today. Therefore, future research should use a recent data benchmark for evaluation. We investigated three ITSs and two projects from each of them. We propose that future research should consider GitHub for DBRD since the existing approaches do not perform as well on GitHub as they do on Bugzilla and Jira. Taking a step ahead, we also compared industry tools like Bugzilla's `FTS` and `VSCodeBot`. We observe that although some tools proposed in research work perform better, `FTS` can serve as a strong baseline for comparison. Furthermore, a DBRD technique, `REP`, proposed in 2011, outperforms advanced deep learning-based techniques that are recently proposed and should also be used as a strong baseline.

In the follow-up work, we focus on the task of DBRD on projects with a typical number of bug reports. We investigated how to combine the advantages of both the traditional DBRD approach and bLLMs and proposed Cupid. Cupid leverages ChatGPT to identify keywords as the input of the state-of-the-art traditional DBRD approach REP. We conduct a comprehensive evaluation on three datasets and compare Cupid with three baselines. The experimental results show that Cupid outperforms the state-of-the-art DBRD techniques in terms of RR@10 on all the datasets. Particularly, Cupid achieves high RR@10 scores ranging from 0.59 to 0.67 on all the datasets investigated.

Chapter 4

Pull Request Title Generation

Contributors often neglect to write a PR description, the role of the PR titles is significant. For example, 219,909 PRs (16.4%) do not have descriptions among our collected 1,341,790 PRs. Another common case is that PRs are displayed in a list view by default so that only the title and other metadata (e.g., author name, tags, and PR ID) are available. Without a PR description, a high-quality title becomes more important for readers to understand the intention of a PR. There are other ways to figure out what a PR is about, such as direct interaction with the PR's owner or checking the details like commit messages and linked issues. However, these are certainly inefficient for software maintenance.

In addition, PRs usually have a fast turnaround: they are either processed fast or left open for a long time without merging to the main branch [67]. In large projects, it is challenging for integrators to handle a high volume of incoming PRs [69]. Prior research [72] on BRs has shown that well-written BRs are more likely to gain the triager's attention and influence on deciding whether a bug gets fixed. Similarly, the quality of PR, such as the length of the title and description, significantly impacts PR evaluation latency [176]. Intuitively, PR titles serve as the first criterion for integrators to decide whether certain PRs are relevant to

their expertise, which can potentially speed up the review process. Nowadays, the quality of PR descriptions has gained more attention in practice: many Git service providers support software maintainers to use templates to improve the quality of PR descriptions.[1] However, there is no emphasis on helping developers compose high-quality PR titles.

To fill in this gap, we aim for the automatic generation of PR titles to compose accurate and succinct PR titles. We formulate the automatic PR title generation task as a one-sentence summarization task. Its goal is to produce a concise and informative target sequence of tokens (*title*), which is a summary of the given *source* sequence of tokens.

As no prior work has been devoted to automatic PR title generation, we comprehensively evaluated the effectiveness of five state-of-the-art summarization methods on this task, including both general-purpose and domain-specific summarization methods. Some of these methods are extractive (i.e., they extract the sentences from the source sequence and then concatenate them to form the target sequence), while others are abstractive (i.e., they generate the target sequence with sentences that are different from the original sentences in the source sequence). For *general-purpose* summarization methods, we have identified three approaches: BERTSumExt [106], BART [94], and Text-To-Text Transfer Transformer (T5) [132]. BERTSumExt [106] is an extractive summarization method that utilizes the pre-trained bidirectional encoder representations from Transformers (BERT) [55]. BART [94] and T5 [132] are two large pre-trained Transformer-based architectures, which can be used for abstractive summarization. For *domain-specific* summarization methods, we evaluate two methods, i.e., PRSummarizer [108] originally designed for PR description generation and iTAPE [46] initially designed for issue title generation. Another related work is by Liu et al. [103] that proposed a Stack Overflow title generation approach; However, the underlying model of their method is T5, which is already included in the general-purpose summarization method.

[1] GitHub documentation about issue and pull request templates

Specifically, in the work, we would like to answer two RQs to understand the performance of different methods on the PR title generation task:

RQ1: *In terms of automatic evaluation, how do different methods perform on the PR title generation task?*

RQ2: *To what extent can the best-performing approaches automatically generate PR titles as developers would do?*

Due to the lack of a suitable dataset, we construct a dataset named PRTiger (Pull Request Title Generation), which is the first dataset that can be leveraged for PR title generation. Our focus is to help contributors compose non-trivial PR titles, which aims to help integrators grasp the main changes made in the PRs. We identified and applied several rules to keep the PR titles that suit the use scenario. In the end, we have 43,816 PR titles belonging to 495 GitHub repositories in PRTiger. The source sequence in the dataset is the concatenation of the PR description, commit messages, and linked issue title, with an average length of 114 words. The *target* sequence is the corresponding PR title, with an average length of 7 words.

ROUGE metrics [100], the standard metrics for automatic evaluation of summarization technique effectiveness, are adopted to evaluate model performance. We also conducted a manual evaluation by inviting three evaluators. For each sample, the evaluators were asked to score three titles by reading the source sequence. The titles generated by the automatic methods and the original human-written titles were randomly shuffled. Evaluation criteria include *correctness*, *naturalness*, and *comprehensibility*. The results suggest that BART outperforms the other techniques in automatic and manual evaluation.

4.1 Benchmarking Dataset Building

As there is no existing dataset for PR title generation, we decided to build the first benchmark dataset for this task. In this work, we experimented with GitHub data due to its popularity. Although Liu et al. [108] shared a dataset for the PR description generation task, they did not include the titles of PRs in their dataset. In addition, their dataset only contains engineered Java projects, which may lack diversity. Therefore, a new dataset containing PR titles gathered from various programming language repositories is needed. We first collected the data; next, we filtered out PRs which did not belong to the usage scenario we focused on in this work. Then, we filtered out the content from PRs which do not help to generate accurate and succinct PR titles.

4.1.1 Data Collection

To collect PR data from GitHub, we first got 7 repository lists: Top-100 most-starred and Top-100 most-forked repositories (regardless of programming language); Top-100 most-starred repositories that are written primarily in one of the following languages: JavaScript, Python, Java, C, and C++.[2] The number of stars and the number of forks are two different metrics in GitHub: The number of stars means how many people click on the repository to show their appreciation. A fork is a copy of a repository and the number of forks indicates how many people copied this project. The Top-100 most-starred and most-forked repositories cover a wide range of programming languages, e.g., Shell (`ohmyzsh/ohmyzsh`), Go (`kubernetes/kubernetes`), and TypeScript (`microsoft/vscode`). Given the 7 lists have common repositories, after removing redundancies, in total we crawled 578 distinct repositories. We collected the PRs from each repository using GitHub

[2]The detailed repository lists are available here: https://github.com/EvanLi/Github-Ranking/blob/f4cf5694eaed7ad1ee59425d7c0dcf9f3e8511f9/Top100

Table 4.1: Our trivial PR title patterns, and example PR titles from the collected dataset.

Starts with (lowercased)	**Example Title**
automated cherry pick of	*Automated cherry pick of #12022 #13148 upstream release 1.0*
merge to	*Merge to live part 2 on 4-27*
revert	*Revert "Add log_only to debug messages"*
rolling up	*Rolling up changes to staging from master*
rolling down	*Rolling down changes from staging to master*
roll engine	*Roll engine dart roll 20180920*
rollup of	*Rollup of 5 pull requests*
roll plugins	*Roll Plugins from 6d8ea78c5da1 to 361567b9189c (4 revisions)*
update live with current master	*Update live with current master*

GraphQL API.[3] For each repository, we only kept the merged PRs published until December 31, 2021. Given a merged PR published before 2022, we retrieved its title, description, commit messages, and linked issues. In total, we collected 1,341,790 merged PRs from the 578 GitHub repositories.

4.1.2 Data Preprocessing

Selecting PRs. For each PR, to better simulate the scenario when a contributor opens a new PR, we first removed the commits submitted after the PR was created. Then, following Liu et al. [108], we filtered out the PRs which have less than two commits or more than 20 commits. As Liu et al. pointed out, we can directly use the commit message as the PR title if a PR only contains one commit, and a PR with too many commits is usually used for synchronization purpose instead of being a typical contribution from developers. We also removed the PRs (1) containing non-ASCII characters; (2) authored by bots. In addition, we also filtered out the PRs which contain *trivial* titles, where automatic methods for generating PR titles are not needed. We mainly identified the following four types of

[3] https://docs.github.com/en/graphql

Table 4.2: Data statistics on our collected pull requests

With < 2 or > 20 commits	With non-ASCII characters	Authored by bot	With trivial titles
1,147,734	16,396	1,642	111,780

PRs Left	Total PRs Collected
51,753	1,341,790

trivial titles: (1) *Recurrent titles.* Table 4.1 shows the templates which occurred prominently in our collected PRs. If any PR title starts with these patterns, we exclude it. (2) *Too short or too long titles.* Following the dataset building rules used by iTAPE [46], we excluded the titles with less than 5 words or more than 15 words. Similar to Chen et al. [46], we observe that PR titles having 5-15 words are of reasonably appropriate length to precisely and succinctly describe key ideas. (3) *Titles with limited overlap with the source sequence.* If 20% of the words in the title were not present in the source sequence, we considered the title not to be a good summary of the source sequence and therefore excluded it from the dataset. (4) *Titles were copied from the source sequence.* We first lowercased both the title and the source sequence. If the title can be exactly matched to the description or concatenation of the commit messages, we removed the PR from the dataset as we consider this PR as a bad example to train the model.

After selecting PRs based on the above criteria, we have 51,753 PRs left in the dataset. Table 4.2 shows the statistics of our collected PRs.

Cleaning the selected PRs. We followed iTAPE [46] to remove tags in the PR titles. We also followed PRSummarizer [108] to (1) remove checklists in the source sequence; and (2) remove identifiers in both source and target sequence. We also added extra steps: (1) Removing PR templates in the source sequence. We first queried through the GitHub API to find out whether a repository provided a PR template for contributors to compose a PR description. If there was a PR template, we saved the template string. In our dataset, 214 out of 495 repositories provided a PR template at the time we called the GitHub API. To

Table 4.3: Data statistics on different splits

	Train		Validation		Test	
	Source	**Target**	**Source**	**Target**	**Source**	**Target**
Instance #	35,052	35,052	4,382	4,382	4,382	4,382
Avg Word #	114	7	114	7	112	7

remove the template information from each PR, we first split the PR description into lines. Then, for each line in the source sequence, if it can be matched exactly to the template string, we removed it. Otherwise, we kept the lines. By doing this, we reduced the noise in the source sequence. (2) Removing automatically generated commit messages in the source sequence. We observed that many commit messages only convey the *merge branch* information, e.g., *Merge branch 'master' into tst-indexing-16*. We used the following four regular expressions to remove the automatically generated commit messages: (1) `merge .*? branch .*? into` (2) `merge branch .*? into` (3) `merge pull request \#` and (4) `merge branch \'`.

After applying the pre-processing steps to our data, we further excluded PRs which have less than 30 words or more than 1,000 words. In the end, we have a dataset *PRTiger* consisting of 43,816 PRs for experiments. We split PRTiger into training, validation, and test sets with a ratio of 8:1:1. Table 4.3 shows the number of instances and the average word count in the train, validation, and test set, respectively.

4.2 Summarization Methods

In this section, we elaborate on the summarization methods evaluated in this work. Summarization methods can be broadly categorized into two groups, i.e., *extractive methods* and *abstractive methods* [24]. An *extractive* summarization method extracts sentences from

the source sequence and then concatenates them to form the target sequence. In contrast, an *abstractive* summarization method represents the source sequence in an intermediate representation. It then generates the target sequence with sentences that are different from the original sentences in the source sequence [56]. We experimented with state-of-the-art extractive and abstractive methods. Their details can be found as follows:

4.2.1 Extractive Methods

We experimented with two extractive summarization methods, i.e., oracle extraction and BertSumExt [106].

Oracle Extraction: This is not a *real* approach. Instead, it can be viewed as an upper bound of extractive summarization methods or serve as a measure to gauge the capability of other extractive summarization methods. Oracle extraction scores have been commonly used in summarization literature for comparison purposes [41, 178]. It may have different variants depending on the specific task setting. In our work, oracle extraction selects the sentence from the source sequence that generates the highest ROUGE-2 F1-score compared to the original title. We first split the PR description, commit messages, and issue titles into sentences. Next, we computed the ROUGE-2 F1-score of each sentence with the original PR title. We selected the sentence with the highest ROUGE-2 F1-score as the generated title by this method. Since oracle extraction needs the original title as a reference, it cannot be applied in practice. However, it can be used for comparison to understand other extractive methods' performance.

BertSumExt [106]: BertSumExt is built on top of BERT-based encoder by stacking several inter-sentence Transformer layers. Specifically, for each sentence in the source sequence, BertSumExt represents each sentence with the vector of the *i*-th `[CLS]` symbol from the top layer. Then, several inter-sentence Transformer layers are then stacked on top of BERT

outputs, to capture document-level features for extracting sentences as the summary. BertSumExt achieves better results with less requirement on the model, compared to other models that enhance the summarization via the copy mechanism [70], reinforcement learning [127], and multiple communicating encoders [44]. Identical to what we did in oracle extraction, we also split PR description, commit messages, and issue titles into sentences. BertSumExt will then give a score to each sentence based on the suitability of becoming a summary. BertSumExt was initially evaluated on three single-document news summarization datasets, which could be summarized in a few sentences. In the original implementation, BertSumExt chooses the sentences with the Top-3 highest scores as a summary. As the PR title generation task was formulated as a one-sentence summarization task, we take the Top-1 sentence as the generated PR title.

4.2.2 Abstractive Methods

The two most similar works are identified, i.e., PRSummarizer [108] and iTAPE [46]. As they did not directly choose the sentence from the source sequence, we also categorized them into abstractive methods. Besides, we applied BART [94] and T5 [132], which are state-of-the-art method for text summarization. (we put the exact model version of the Hugging Face `transformers` library that we use [4] in parentheses):

PRSummarizer [108]: This text summarization model was designed to automatically generate PR descriptions from the commits submitted with the corresponding PRs, which is a sequence-to-sequence (Seq2seq) learning task. The underlying model of PRSummarizer is the attentional encoder-decoder model [29]. Besides, PRSummarizer can handle two unique challenges, i.e., out-of-vocabulary (OOV) words and the gap between the training loss function of Seq2seq models and the discrete evaluation metric ROUGE. Especially,

[4] https://huggingface.co/models

PRSummarizer copes with OOV words by integrating the pointer generator [147]. The pointer generator either selects a token from the fixed vocabulary or it will copy one token from the source sequence at each decoding step. To minimize the gap between the loss function and the ROUGE metrics, PRSummarizer also leverages a reinforcement learning (RL) technique named self-critical sequence training [135] and adopts a particular loss function named RL loss [127].

iTAPE [46]: iTAPE uses a Seq2seq-based model to generate the issue title using the issue body. Specifically, iTAPE adopts the attentional RNN encoder-decoder model. In addition, to help the model process the low-frequency human-named tokens effectively, iTAPE first inserts additional "tag tokens" before and after each human-named token, i.e., identifiers and version numbers. These tag tokens are added to the issue body to indicate their latent semantic meanings. Furthermore, iTAPE adopts a copy mechanism [70], which allows the model to copy tokens from the input sequence. The underlying architecture is the pointer-generator [147], a commonly used abstractive summarization approach before the dominant usage of pre-trained models.

BART [94] (`facebook/bart-base`): is a Seq2seq autoencoder based on a standard Transformer [163] architecture. The pre-training process of BART consists of two stages: (1) corrupt the input text by using an arbitrary noising function and (2) a Seq2seq model is learned to reconstruct the original text by minimizing the cross-entropy loss between the decoder output and the original sequence. A number of noising approaches are evaluated in the BART paper. The best performance is achieved by adopting both the noising methods, i.e., (1) randomly shuffling the order of the original sentences, and (2) applying an in-filling scheme, where a single mask token is used to replace arbitrary length spans of text (including zero length). BART was pre-trained with the same data as RoBERTa [107], i.e., 160GB of news, books, stories, and web text. BART achieves new state-of-the-art results on several text generation tasks, including abstractive dialogue, question answering,

and summarization tasks.

T5 [132] (`t5-small`): T5 is a pre-trained language model which aims to convert all NLP tasks into a unified text-to-text-format where the input and output are always text strings. The advantage of this T5 text-to-text framework is that we can use the same model, loss function, and hyper-parameters on any NLP task. T5 is also based on the Transformer architecture. Similar to BART, T5 was pre-trained on a masked language modeling objective: contiguous spans of token in the input sequence are replaced with a mask token and the model is trained to reconstruct the masked-out tokens. Unlike BART, T5 was pre-trained with the Colossal Clean Crawled Corpus (C4) dataset, which consists of 750GB of English text from the public Common Crawl web scrape. T5 was reported to achieve state-of-the-art results on many benchmarks including summarization, question answering, and text classification.

4.3 Study Design

This section describes the relevant design and settings of our study. We list two research questions and the evaluation metrics, and briefly describe the implementation details.

4.3.1 Research Questions

We would like to empirically evaluate the performance of different approaches on the automatic PR title generation task. The study aims to answer the following RQs:

RQ1: *In terms of automatic evaluation, how do different methods perform on the PR title generation task?* Although there is no prior work on automatically generating PR titles, we choose the approaches from the two closest works [108, 46] as the baselines. We used the

implementations of these two approaches in our task. For comparison, we also evaluated the state-of-the-art general-purpose extractive and abstractive summarization techniques.

RQ2: *To what extent can the best-performing approaches automatically generate PR titles as developers would do?* Other than providing the results on automatic evaluation, we also conducted a manual evaluation. Given the expense of running a manual evaluation, we only evaluated the two best-performing methods on the automatic evaluation. We invited three annotators who are not an author of this work.

4.3.2 Evaluation Metrics

Automatic Evaluation Following the prior works [108, 46], we use Recall-Oriented Understudy for Gisting Evaluation (ROUGE) [100] to measure the quality of generated summaries for the summarization task. ROUGE-N measures the overlap of n-grams [100] between the model-generated summary and the reference summary. The formulas to calculate ROUGE-N can be shown as follows:

$$R_{ROUGE-N} = \frac{Count(overlapped\_N\_grams)}{Count(N\_grams\_in\_reference\_summary)}$$

$$P_{ROUGE-N} = \frac{Count(overlapped\_N\_grams)}{Count(N\_grams\_in\_generated\_summary)}$$

$$F1_{ROUGE-N} = 2 \times \frac{R_{ROUGE-N} \times P_{ROUGE-N}}{R_{ROUGE-N} + P_{ROUGE-N}}$$

As the variable names suggest, R (recall) measures the percentage of the N-grams in the reference summary that the generated summary has covered, and P (precision) presents the percentage of N-grams in the generated summary that is, in fact, relevant or needed. F1-score of the ROUGE scores is used to represent and give an equal importance between recall and precision. In this task, we report the precision, recall, and F1-score of ROUGE-N

Table 4.4: An example PR with the original title and the titles generated by BART and T5. https://github.com/grpc/grpc/pull/1655

Source	<desc> this fixes #1551. servers now respond to unparseable arguments with the invalid_argument status and the text of the deserialization error, instead of crashing. </desc> <cmt> added failing tests for server bad argument handling </cmt> <cmt> fixed server to handle invalid arguments without breaking </cmt> <iss> node rpc server cannot recover from malformed requests </iss>
Original Title	handle invalid arguments sent to the node server
BART	fix server bad argument handling
T5	fix server to handle invalid arguments without breaking node

(N=1,2) and ROUGE-L from each method. ROUGE-1, ROUGE-2, and ROUGE-L are commonly used in the literature to understand the summary quality [108, 46]. ROUGE-1 and ROUGE-2 measure the overlap of uni-grams (1-grams) and bi-grams (2-grams), respectively. A uni-gram consists of a single word, while a bi-gram consists of two consecutive words. Instead of n-grams, ROUGE-L measures the longest common subsequence between the reference summary and the generated summary. We treat F1-score of ROUGE as the main summarization performance measure. We first get the generated summaries from each approach. For the ROUGE scores calculation, we adopt the metric implemented in Hugging Face `datasets` library [95].

Manual Evaluation In addition to automatic evaluation, we also conducted a manual evaluation to understand better and evaluate the quality of titles generated by different approaches. Since ROUGE scores are calculated based on the overlap of n-grams, the generated summaries may be semantically incorrect, even with very high ROUGE scores. This is due to the ROUGE scores limitation which only measures the lexical similarity between two sentences. Hence, it cannot gauge the comprehensibility of the summaries generated by the model. [162] Thus, we sampled 150 PRs from the test set. With this sample size, we can maintain a confidence level of 92% with an 8% margin of error. Three evaluators were invited to give the quality scores to the PR titles generated by the two best

approaches and the original titles written by developers. All evaluators have more than 6-year experience in programming and more than 5-year experience using GitHub. They have a basic understanding of how pull-based software development works (i.e., they are experienced in making changes, pushing commits, writing issue reports, and opening PRs).

The techniques used to generate titles are hidden from the evaluators; they cannot judge based on the bias of knowing the authorship. For each sample, evaluators were provided with the *source* sequence along with three *titles*: two of the titles are generated by the two best-performing approaches, i.e., BART and T5, and the rest is the original title. We randomly shuffled the order of the three titles. To help the evaluators read the source sequence clearly, we use `<desc></desc>`, `<cmt></cmt>`, and `<iss></iss>` to enclose description, commit messages, and linked issue titles, respectively. One sample source sequence can be seen in Table 4.4. Evaluators were required to read through the *source* sequence and the three titles. They were asked to score the three titles (1 - very poor; 5 - very good) with regards to the following aspects:

- **Correctness:** To which extent, do you think the title *correctly* summarize the source sequence?
- **Naturalness:** To which extent, do you think the title is resembling a human-written title?
- **Comprehensibility:** To which extent, do you think the title is easy to *understand*?

Additionally, they were also required to rank the three titles. Their personal preference for these titles is not necessarily based on the three criteria listed above. If the titles are the same, they can rank two titles with the same rank. Otherwise, they must give different ranks for each title.

Table 4.5: Automatic evaluation Results on the test dataset

Approach	Avg. Length	ROUGE-1			ROUGE-2			ROUGE-L		
		P	R	F1	P	R	F1	P	R	F1
Extractive										
– **Oracle Extraction**	13	47.61	55.74	46.97	32.51	34.04	30.25	44.88	51.33	43.91
– **BertSumExt**	13	36.18	44.95	37.64	18.18	21.71	18.48	32.76	40.28	33.94
Abstractive										
– **PRSummarizer**	6	41.4	36.68	37.91	20.06	17.26	17.99	38.22	33.83	34.98
– **iTAPE**	8	31.63	34.62	32.23	12.67	13.98	12.91	28.71	31.51	29.31
– **BART**	7	50.03	46.98	47.22	27.15	24.96	25.27	45.71	42.85	43.12
– **T5**	7	44.88	42.15	42.06	23.22	21.3	21.39	41.09	38.49	38.46

4.3.3 Implementation Details

We run all the experiments with NVIDIA Tesla V100 GPUs. For iTAPE, we preprocessed the identifier and version numbers with the scripts provided by the authors.[5] For PRSummarizer, we changed the vocabulary size from 50k to 200k to handle the OOV issue. Given BART (`BART-base`, which contains 140 millon parameters[6]) has more parameters than T5 (`T5-small`, which contains 60 million parameters[7]), we run BART with the batch size of 4; while 8 for T5. All the remaining hyper-parameters were left as the default values. The detailed default values can be found in our replication package.[8]

Table 4.6: Performance comparison with other summarization tasks: issue title generation (Chen et al. [13]), PR description generation (Liu et al. [12]), and lay language summarization of biomedical scientific reviews (Guo et al. [30])

Method	ROUGE-1 F1	ROUGE-2 F1	ROUGE-L F1
BART in our work	47.22	25.27	43.12
Chen et al. [46]	31.36	13.12	27.79
Liu et al. [108]	34.15	22.38	32.41
Guo et al. [73]	53.02	22.06	50.24

4.4 Study Results

4.4.1 RQ1: Comparison on Automatic Evaluation

To investigate how different approaches perform on the PRTiger dataset, we firstly analyze model performance with ROUGE metrics. Table 4.5 shows the ROUGE scores from the six approaches.

Firstly, looking at the length of generated titles, we can observe that the extractive approaches produce longer titles than the abstractive approaches. The average length of the titles generated by extractive approaches highly relies on the length of sentences selected from the dataset. Although we set the extractive approaches to select a single sentence as a title, they select longer titles. All the abstractive methods generate titles with an average length of 6-8 words. The length range is similar to the average length across all the data splits (See Table 4.3). It indicates that abstractive methods generate titles with an appropriate length, as it is capable of generating titles with an average length similar to

[5] https://github.com/imcsq/iTAPE

[6] https://github.com/pytorch/fairseq/blob/main/examples/bart/README.md#pre-trained-models

[7] https://github.com/google-research/text-to-text-transfer-transformer#released-model-checkpoints

[8] https://github.com/soarsmu/PRTiger

the average length of the original titles.

Serving as the upper bound of extractive summarization approaches, Oracle Extraction gives the highest ROUGE-1, ROUGE-2, and ROUGE-L F1-scores since it relies on the original title. In comparison, BertSumExt gave ROUGE-1, ROUGE-2, and ROUGE-L F1-score of 37.64, 18.48, and 33.94. These scores were 20%, 38.9%, and 22.7% lower than the Oracle Extraction. It suggests that, in practice, PR titles have a considerable amount of overlap with the source sequence. Yet, BertSumExt is not capable of selecting the correct sentence every time.

Among the four abstractive approaches, BART and T5 achieved better performance than the two other abstractive approaches. It demonstrates the power of pre-training in the PR title generation. BART and T5 were both pre-trained with large general-purpose corpora. Unlike these two approaches, PRSummarizer and iTAPE were trained solely on the PRTiger data, where PRSummarizer produces a better performance than iTAPE. Considering PRSummarizer was originally proposed for PR description generation, the data they used to evaluate their approach has similar characteristics as PRTiger. In addition, both PRSummarizer and iTAPE adopt the pointer generator [147]. It suggests the importance of RL-loss in PRSummarizer, which is used to minimize the gap between the loss function and the ROUGE metrics.

Comparing extractive and abstractive methods, BART shows an on-par performance as the Oracle Extraction, which indicates that BART is capable of capturing key points from the source sequence and generating precise PR titles. BART outperforms the second-best approach T5 by 12.2%, 18.1%, and 12.1%, in terms of ROUGE-1, ROUGE-2, and ROUGE-L F1-scores, respectively. Although T5 gives worse performance than BART, it still outperforms the other approaches, i.e., PRSummarizer, by 10.9%, 18.9%, and 10% with regards to ROUGE-1, ROUGE-2, and ROUGE-L F1-scores, respectively. Interestingly, BertSumExt gives an on-par performance as PRSummarizer and it outperforms iTAPE.

Given the words in the PR, titles are not necessarily contained in the source sequence, the PR title generation task is naturally an abstractive summarization task. However, as an extractive approach, BertSumExt shows relatively good performance. It indicates that, in practice, developers may draft titles based on existing sentences, either in PR description or commit messages.

Other than comparing different approaches on the automatic PR title generation task solely, we show the ROUGE F1-scores from other related tasks. Chen et al. [46] work on the issue title generation task, where the proposed approach is named iTAPE and is evaluated in our work. Liu et al. [108] work on the PR description generation task, where the proposed approach is named PRSummarizer and is evaluated in our work as well. Guo et al. [73] work on automated generation of lay language summaries of biomedical scientific reviews. The ROUGE F1-scores from the best performing method of these three tasks are present in Table 4.6. According to the result, we can find that BART in our work achieves a comparable level of performance on the automatic PR title generation task.

Automatic Evaluation The fine-tuned BART and T5 outperform all the other approaches. BertSumExt is on par with the two existing approaches (PRSummarizier and iTAPE). The best-performing approach, BART, outperforms the second-best approach by 12.2%, 18.1%, and 12.1%, in terms of ROUGE-1, ROUGE-2, and ROUGE-L F1-scores.

4.4.2 RQ2: Comparison on Manual Evaluation

Figure 4.1 shows the average scores regarding three aspects from the three evaluators. BART is better than T5 in the three aspects, which is in line with the automatic evaluation results with ROUGE metrics. Surprisingly, we can see that BART and T5 show higher scores than the original titles in all three perspectives. It indicates that the titles generated by automatic methods are more acknowledged than the titles written by PR authors.

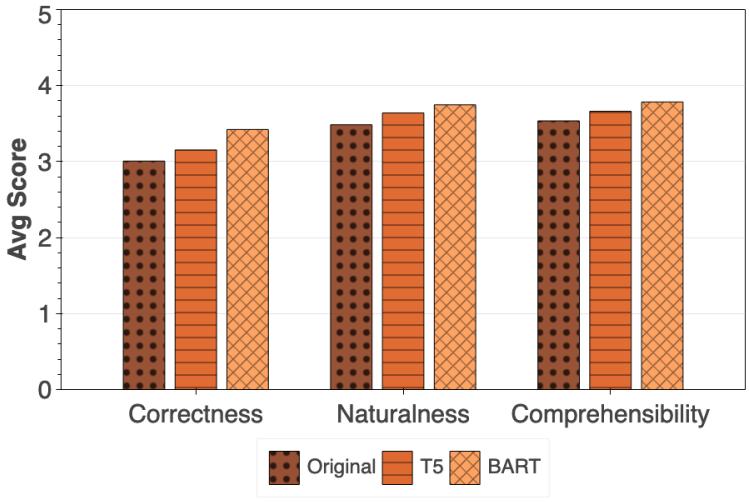


Figure 4.1: Average scores from three evaluators

Across the three aspects, we can find that all three approaches receive the lowest average scores of *correctness*. They achieve slightly higher average scores regarding *comprehensibility* than *naturalness*. These results show that although PR titles read like humans wrote them and are easy to understand, they may not be correct enough from the evaluators' perspectives.

Figure 4.2 shows that for each rank, the percentage of different approaches. Although the ranking is a relative order, it is allowed to be subjective and solely based on the evaluators' personal preference. For the best ones, we can find that BART has been listed as the best one the most times. T5 comes the second. Again, surprisingly, the original title is less preferred.

While a larger-scale study is required, our work provides preliminary evidence that automatically generated PR titles are readable and comprehensible to PR readers, i.e., have higher scores in terms of correctness, naturalness, and comprehensibility. In the sampled PR titles, automatically generated PR titles are preferred over the original titles in most cases.

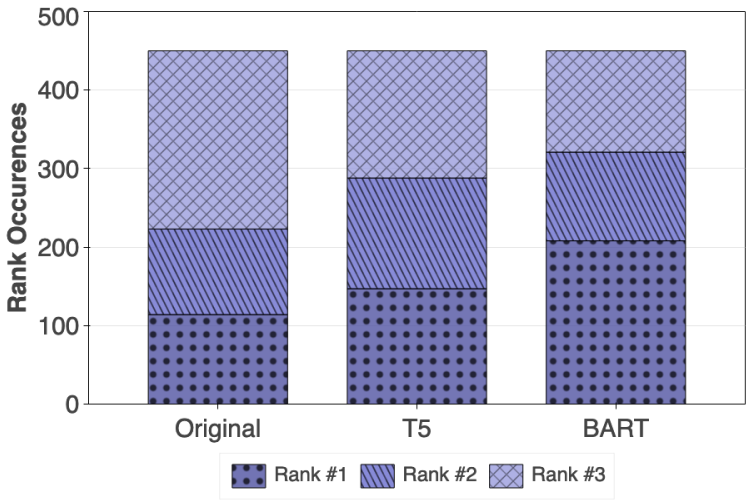


Figure 4.2: Rank occurrences for the original titles and the titles generated by BART and T5.

Manual Evaluation The PR titles generated by BART gained the highest scores on correctness, naturalness, and comprehensibility. T5 performs the second. They are both preferred more than the original titles.

4.5 Discussion

4.5.1 Qualitative Analysis

Composing PR titles is a non-trivial task, as it requires both summarization ability and domain knowledge. Moreover, PR titles are generally short. Here, we would like to seek a qualitative understanding of the performance difference between different approaches.

Automatic Evaluation. In Table 4.7, we showcase one example of titles produced by each approach and the original title along with the source sequence. Note the source sequence is the cleaned version used for the models to generate titles. We use boldface to highlight the common part between the text and the original title. All the approaches

Table 4.7: An example PR with the original title and the generated titles from five approaches: https://github.com/iluwatar/java-design-patterns/pull/1066

Source Sequence	reduces **checkstyle errors for** patterns: **api-gateway lazy-loading leader-election** changes involved java docs reordering imports indentations line length issues reduces **checkstyle errors** in **lazy-loading** reduces **checkstyle errors** in **leader-election** reduces **checkstyle errors** in **api-gateway**
Original Title	**resolves checkstyle errors for api-gateway, lazy-loading, leader-election**
BART	**resolves checkstyle errors for api-gateway lazy-loading leader-election**
T5	reduces **checkstyle errors for** patterns
BertSumExt	reduces **checkstyle errors for** patterns : **api - gateway lazy - loading leader - election** changes involved java docs reordering imports indentations line length issues
PRSummarizer	reduces **checkstyle errors for** patterns: **api-gateway**
iTAPE	resolves **checkstyle errors for** patterns : **api-gateway lazy-loading**

can correctly generate `checkstyle errors for`, which occurs several times in the source sequence. However, all the approaches except BART generate the first word as `reduces`, and they also generate `pattern`. It is natural, as the first sentence in the source sequence contains `reduces` and `pattern` as well. It is interesting to see that BART does not directly take the first sentence. Besides, the body does not even have the word `resolve`. We checked the dataset and found that several PR titles in the same repository (`iluwater/java-design-pattern`) followed the same name style: `resolves checkstyle errors for`. The following text differs among PRs. From this example, we can find that BART can learn this title style instead of simply choosing the first sentence. BART does not only use exactly the same words from the source sequence. Instead, it could correctly generate the words which are not present in the source sequence. It shows BART has the promising ability to be adopted in the automatic PR title generation task.

Manual Evaluation. In Table 4.4, we show an example from our sampled 150 PRs. All the evaluators indicate their preference among the three titles as: BART > T5 > Original. Firstly, this PR contains description, two commit messages and linked issue. The original

title uses `handle`, which only appears in one of the commit messages. Besides, the phrase `sent to` does not exist in the source sequence. In comparison, the generated title from BART and T5 used `fix`, which occur twice in the source sequence and all the words in the generated titles are present in the source sequence. The title generated by BART is shorter and summarizes the source sequence well, while the title generated by T5 is longer and covers more words from the source sequence.

We investigated PR titles with high comprehensibility scores (i.e., all evaluators gave a score of >=4 for comprehensibility). We found that a PR title is considered to be more comprehensible when: (1) It covers multiple sources of information, e.g., sentences in the PR description and commit messages. (2) It explicitly states that it fixes an issue or adds a feature (e.g., usually starts with the word "fixed" or "added"). We also investigated PR titles with low comprehensibility scores (i.e., all evaluators gave a score <=3 for comprehensibility). We found that a PR title is considered to be less comprehensible when: (1) the PR contains many commits but the PR title only takes information from a few commit messages. (2) The PR title has little connection with the PR description.

4.5.2 Threats to Validity

Threats to internal validity relate to the experimental bias. Following the prior works in summarization studies [108, 46], we adopted both automatic evaluation (i.e., ROUGE metrics) and manual evaluation. Like other manual-involved evaluations, our experimental results may be biased or inconsistent. To mitigate this issue, we recruited 3 evaluators with 6+ years of computer programming and 5+ years of experience in using GitHub. They are familiar with the issue and code review mechanism in GitHub.

Moreover, during the dataset building process, although we have performed the selection heuristics, the remaining PR titles that we use as ground truth for automatic evaluation

may not be the ideal ones. We use them as proxies of the ideal reference titles for scalability reasons. There are a total of 43,816 titles in our experimental dataset and crafting the best possible reference titles for all of them manually is not practically feasible. We address this limitation of the automatic evaluation by manual evaluation; each has its own limitations: automatic (quality of reference titles) vs. manual (limited number of titles are considered).

Threats to external validity relate to whether our findings can be generalized to other datasets. To alleviate the external threats, we consider repositories from different programming languages diverse. We also considered two metrics in GitHub: Top-100 most-starred and Top-100 most-forked repositories. Although we only included the repositories from GitHub, we do not identify a large difference between repositories from GitHub and those from other git service providers [34]. Therefore, we consider the external threat to be minimal.

4.6 Summary

In this work, we propose the task of automatically generating PR titles. To facilitate the research on this task, we constructed a dataset named PRTiger, which consists of 43,816 PRs from 495 GitHub repositories. We conducted both automatic and manual evaluations on the state-of-the-art summarization approaches for the automatic PR title generation task. The experimental results indicate that BART is the most capable technique for generating satisfactory PR titles with ROUGE-1, ROUGE-2, and ROUGE-L F1-scores of 47.22, 25.27, and 43.12, respectively. The manual evaluation also shows that the titles generated by BART are preferred.

We believe that our work opens up many interesting research opportunities. To name a few, one possible research direction is to consider the characteristics of PR data to propose

a domain-specific pre-trained model. Domain-specific models are promising, such as BERTweet [119], which is pre-trained on English tweets, outperforms the strong baseline $\text{RoBERTa}_{\text{base}}$ [107] on three Tweet NLP tasks. Additionally, to further improve the PR title generation performance, another direction is to leverage the hierarchical code structure information from code changes.

Chapter 5

Related Works

This chapter covers the work which is most relevant to the dissertation.

5.1 Boosting SA4SE Accuracy

Previous research has shown that emotions influence work outcomes and dynamics, such as task quality, productivity, creativity, group rapport, user focus, and job satisfaction (c.f. [52, 124]). On the other side of the coin, work processes and outcomes influence developer emotions (c.f. [140]). Much research has investigated aspects of this two-way relationship between developers' work and their emotions.

One line of work that has attracted much research interest is the SA of software artifacts, such as BRs and commit comments. For example, Guzman et al. [74] studied sentiments in commit comments in GitHub to analyze the social factors affecting software development. Villarroel et al. [164] mined emotional information from mobile APP reviews to support the release planning activity. To further analyze the impact of negative code review comments, Ahmed et al. [22] developed a code-review specific SA tool. Fine-grained emotions have

also been studied. Gachechiladze et al. [61] focus on the automatic identification of anger direction (anger towards self, others, and objects) in a collaborative software development environment. They found that all of the anger directions are present within the comments from Apache issue reports [61].

The progress of SA research in SE has promoted the development of corresponding tools. In the past decades, many techniques have been proposed to improve the effectiveness of identifying sentiments or emotions in the SE domain [47, 48, 82, 76, 22, 42, 43, 77, 118, 81, 35]. Chen et al. [47] propose SEntiMoji, an emoji-powered learning approach for SA in SE. They employ emotional emojis as noisy labels of sentiments and propose a representation-learning approach that uses both Tweets and GitHub posts containing emojis to learn sentiment-aware representations for SE-related texts. In the evaluation, they compare SEntiMoji with four SA4SE tools on sentiment polarity benchmark datasets. The experimental results show that SEntiMoji can significantly improve the performance.

Furthermore, Chen et al. [48] include an additional evaluation of SEntiMoji on the emotion detection task. They also compared it with four existing emotion detection methods, including DEVA [81], EmoTxt [43], MarValous [77], and ESEM-E [118]. The experimental results on the five benchmark datasets covering 10,096 samples for sentiment detection and four benchmark datasets covering 10,595 samples for emotion detection demonstrate that SEntiMoji is effective.

Besides developing a new SA4SE tool, some research aimed at improving SA accuracy by handling existing challenges, such as labeled data scarcity. Imran et al. [76] address the data scarcity problem by automatically creating new training data using a data augmentation technique. They specifically target the data augmentation strategy to improve the performance of emotion recognition by analyzing the types of errors made by popular SE-specific emotion recognition tools. Their results show that when trained with their best augmentation strategy, three existing emotion classification tools, i.e., ESEM-E, EMTk, and

SEntiMoji, received an average improvement of 9.3% in micro-F1 score.

5.2 Empirical Studies in SA4SE

With the proliferation of domain-specific SA4SE tools, a series of empirical investigations have been conducted to illuminate our understanding of this field's progress and challenges [99, 123, 85, 125, 86, 122, 98].

By comparing the performance of general-purpose SA tools in the SE field, Jongeling et al. [85] found that these tools produced inconsistent annotated labels, and they may not necessarily agree with each other. Therefore, they claim a need for SE domain-orientated SA tools. Due to the non-optimal performances of these off-the-shelf SA tools built from the general text, more SE domain-specific SA tools have been introduced.

Novielli et al., in their recent study [122], delve into the critical question of how off-the-shelf, SE-specific SA tools affect the conclusion validity of empirical studies in SE. They begin by replicating two prior studies that explore the role of sentiment in security discussions on GitHub and question-writing on Stack Overflow. Subsequently, they extend these studies by assessing the level of agreement between different SA tools and manual annotations, using a gold standard dataset comprising 600 documents. The experimental findings from this research reveal that when applied out-of-the-box, various SA4SE tools may yield conflicting results at a granular level. Consequently, it becomes imperative to consider platform-specific fine-tuning or retraining to account for differences in platform conventions, jargon, or document lengths.

Obaidi et al. [125] conducted a systematic mapping study to comprehensively examine SA tools developed for or applied in the SE domain. This study summarizes insights drawn from 106 papers published up to December 2020, focusing on six key aspects: (1) the

application domain, (2) the purpose of SA, (3) the datasets used, (4) the approaches for developing SA tools, (5) the utilization of pre-existing tools, and (6) the challenges faced by researchers. Based on their findings, neural networks emerge as the top-performing approach, with BERT identified as the most effective tool.

Beyond the scope of sentiment classification, some researchers have explored broader facets of SA4SE, such as opinion mining. Opinion mining encompasses a wider spectrum of tasks than the sentiment polarity identification typically evaluated in SA4SE studies. It includes SA, subjectivity detection, opinion identification, and joint topic SA. In a comprehensive systematic literature review, Lin et al. (2022) investigated 185 papers on opinion mining in SE [98], shedding light on the diverse research efforts in this area.

Each of these existing empirical studies has contributed valuable insights into the evolving landscape of SA4SE. Notably, our work stands apart from these studies as it introduces the use of bLLMs to this domain for the first time.

5.3 DBRD Techniques and Practitioners' Perception

Many studies have developed DBRD techniques in the past decade. The state-of-the-art approaches and popular ones have been described in Section 3.1.1. Here, we introduce three classic works and a recent study assessing practitioners' perceptions of DBRD.

One of the pioneer studies in DBRD is by Runeson et al. [142]. They extracted textual fields in a BR (summary and description), and converted a BR into a vector of weights following the standard Vector Space Model (VSM) [146]. Duplicates are then identified by comparing the vector representation of an incoming BR to those of existing ones wrt. three well-known similarity metrics: Cosine, Jaccard, and Dice [146]. Wang et al. [165] extended Runeson et al.'s work by considering both natural language text and execution information

(e.g., stack traces). They have shown that the consideration of execution information is beneficial but not many BRs contain such execution information. Sun et al. [153] trained a discriminative model via Support Vector Machine (SVM) to classify whether two BRs are duplicates of one other with a probability. Based on this probability score, they retrieve and rank candidate duplicate BRs.

Zou et al. [195] surveyed 327 practitioners from diverse backgrounds to investigate practitioners' perceptions of DBRD and other automated techniques supporting ITS. They find that DBRD is among the top-3 techniques deemed to be the most valued and find that respondents appreciate these techniques as they can save developers' time, save reporters' time, provide hints for bug fixing, etc.

5.4 Evaluation of DBRD Techniques

There are several studies on evaluating DBRD techniques [133, 160]. Rakha et al. [133] studied the differences between duplicate BRs before and after the introduction of JIT duplicate BR recommendation (JIT feature) in Bugzilla (in 2011). They found that duplicate BRs after 2011 (2012–2015) are less textually similar, have a greater identification delay, and require more discussion to be retrieved as BRs than duplicates before 2011. Their study also demonstrates that when evaluating the data after 2011, the experimental results of prior research would vary. These findings motivated us to experiment on BRs submitted after 2011. Based on their findings, we built the old and recent data both after 2011. Our work differs from their work as we investigate the two 3-year time window data after the JIT feature. We investigate not the impact of JIT DBRD feature introduction in Bugzilla, but rather *age bias*.

Tu et al. [160] also highlighted the fact that the BR attributes, i.e., field values, change

over time. They raise a concern that several DBRD techniques use data from the future while training their models. Their study can be seen dealing with our *state* bias. They just investigate REP [152]'s accuracy from BRs on Bugzilla ITS of Mozilla and Eclipse. In this work, we investigated diverse factors that could affect the performance of DBRD techniques, and not only the *state* bias. Furthermore, we run a statistical test on the accuracy difference between using the initial state and the latest state and show that state bias does not have a statistically significant impact.

5.5 Understanding Pull Requests

Pull-based software development has attracted more and more interest in research. Some works focus on empirically understanding PRs. Gousios et al. conducted two surveys to analyze the work practices and challenges in pull-based development from the integrator's [69] and contributor's [68] perspectives, respectively. In the first piece of work, they found that integrators successfully use PRs to solicit external contributions. Integrators concerned with two major factors in their daily work, i.e., (1) quality: both at the source code level and tests (2) prioritization: typically, they need to manage a large number of contribution requests at the same time. In the latter one, they performed an exploratory investigation of contributors to projects hosted on GitHub. Their findings include but are not limited to, the fact that contributors are very interested in knowing project status for inspiration and to avoid duplicating work, but they are not actively trying to publicize the PRs they are preparing.

Other works are interested in solving PR-related tasks. Yu et at. [177] studied the reviewer recommendation for PRs to improve the PR evaluation process. They found that traditional approaches (e.g., SVM-based) for bug triage are also feasible for PR reviewer recommendations. They also found that combining the social factors (e.g., common in-

terests among developers) and technical factors (e.g., developer's expertise) is efficient in recommending reviewers. Jiang et al. [84] studied the tag recommendation for PRs. They proposed FNNRec, which uses a feed-forward neural network to analyze titles, descriptions, file paths, and contributors to help developers choose tags. These prior works motivate our work to automatically generate high-quality PR titles to benefit both developers and reviewers. Besides, good PR titles can also be helpful for downstream tasks that utilize PR titles.

5.6 Automatic Software Artifact Generation

Automatic software artifact generation has gained emerging interest in SE research. Existing works range from automatically generating release notes [117], BRs [96], to commit messages [173]. Moreno et al. [117] introduced an automatic approach to generate release notes. The proposed approach extracts changes from the source code, summarizes them, and integrates the summary of changes with information from versioning systems and issue trackers. Li et al. [96] proposed an unsupervised approach for BR summarization. The approach leverages an auto-encoder network with evaluation enhancement and predefined field enhancement modules. Xu et al. [173] proposed CODISUM to address the two limitations of the prior commit message generation task, i.e., ignoring the code structure information and suffering from the OOV issue. Specifically, to better learn the code change representations, they first extract both code structure and code semantics from the source code changes and then jointly model these two information sources. Moreover, they adopted a copy mechanism to mitigate the OOV issue. We also mention iTAPE, an issue title generation approach in Section 4.2. Our work and these works are complementary. We all aim to promote automatic generation in SE, while concentrating on different aspects.

5.7 sLLMs for SE

Given the success witnessed in many NLP tasks, researchers in the SE field have started to utilize sLLMs to solve many SE-related tasks as well. There are generally two lines of research on sLLMs for SE. The first focuses on pre-training large-scale domain-specific models in the SE field [88, 71, 58]. One example is BERTOverflow [154], which was $\text{BERT}_{\text{base}}$ pre-trained on Stack Overflow 10-year archive of 152 million sentences and 2.3 billion tokens. It was originally designed to help identify code tokens or software-related named entities that appear with natural language sentences, especially on Stack Overflow. The experimental results on the name entity recognition task on the Stack Overflow dataset show that fine-tuning over BERTOverflow improves F1-score by more than 10 points compared to using the off-the-shelf BERT. Another example is CodeBERT [58], which is a bimodal sLLM for both programming languages and natural languages. The experimental result showed that fine-tuning CodeBERT achieved state-of-the-art performance on two downstream tasks, i.e., natural language code search and code-to-documentation generation. Similar to CodeBERT, there are also many sLLMs that have been pre-trained on programming languages, such as GraphCodeBERT [71] considers the inherent structure of code.

The second one pays attention to fine-tuning sLLMs for boosting the performance in downstream SE tasks [182, 101]. For example, traceability recovery [101], library recognition in tweets [182] and code search [58]. Zhang et al. [182] extensively evaluated a broad set of sLLMs, including both general-purpose and domain-specific ones, to solve this programming library recognition task in tweets. Experimental results show that using sLLMs can outperform the best-performing baseline methods by 5% - 12% in terms of F1-score under within-, cross-, and mixed-library settings. Lin et al. [101] utilized CodeBERT, a variant of BERT pre-trained on source code and documents, to recover links between issues and commits on open-source projects. The results indicate that the architecture that applies

CodeBERT generates trace links more accurately than the state-of-the-art approaches, e.g., RNN. Both these two works demonstrate that sLLMs can benefit the downstream tasks. Similar to these two works, our work exploits sLLMs to boost the performance for SE.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this dissertation, our primary objective is to enhance the support provided to software developers in their daily tasks, with a particular focus on three tasks, ranging from classification and ranking to generation.

Starting with the SA4SE task, this dissertation conducts an extensive empirical study by comparing the effectiveness of sLLMs against existing SA4SE tools. Furthermore, this dissertation revisits this task by performing a comparative analysis between bLLMs and sLLMs, shedding light on the evolving landscape of SA4SE.

In the second study on the DBRD task, we lay the foundation by creating a benchmark and presenting an innovative approach that combines bLLMs and REP. This approach aims to improve the accuracy of DBRD, reducing the duplication of efforts in software development and enhancing the overall bug reporting and triaging process.

Lastly, in the third study on automatic PR title generation, we demonstrate the superiority

of BART in generating PR titles, as validated through both automatic and manual evaluations. The generated PR titles have the potential to enhance the clarity and informativeness of PRs, ultimately streamlining the development process.

6.2 Future Work

This section outlines potential avenues for future work to further advance the research and findings presented in this dissertation.

Leveraging SA for other Downstream Tasks: The results of SA4SE can serve as a valuable resource for many downstream tasks, such as API or library recommendations. By employing SA4SE methods to gauge the sentiment within various platforms, developers, library maintainers, and open-source contributors can make informed decisions. These insights can provide valuable information about library popularity, user satisfaction, and community sentiment, aiding in library selection and improvement prioritization.

Identifying more Fine-grained Emotions: Other than the sentiment polarities investigated in this work, we are also curious about identifying more fine-grained emotions like frustration, excitement, or confusion in SE contexts. Recognizing these subtle emotional undertones can provide a richer understanding of user experiences, developer challenges, and overall system feedback. This line of inquiry holds promise for enhancing communication between developers, improving software documentation, and even improving software products. In the future, we aim to investigate methodologies that can effectively capture these more fine-grained emotions, potentially leading to more empathetic and user-centric software development practices.

Leveraging Multi-Modal Information in BRs: Enhancing the accuracy of DBRD can be achieved by embracing multi-modal information. The inclusion of screenshots and

potentially other media types such as videos can provide richer context for BRs. Instead of treating all the text as pure text, considering multi-modal information can lead to more precise and comprehensive DBRD approaches.

Improving DBRD Efficiency: While existing DBRD techniques have improved detection accuracy, the response time remains a challenge. Future work can focus on achieving real-time BR analysis. This involves developing efficient algorithms, utilizing distributed computing, and optimizing hardware resources to ensure that BRs are processed swiftly. Reducing bottlenecks in the bug triage process will be crucial in meeting the demands of fast-paced software development.

Considering More Information in Generating PR Titles: While the dissertation has predominantly focused on PR descriptions, commit messages, and linked issue titles, there is an opportunity to incorporate additional sources of information. The hierarchical code structure within the code changes associated with a PR can offer significant context. Analyzing how changes fit into the broader code structure, including affected modules, components, or functions, can yield more informative and contextually relevant PR titles.

These future research directions build upon the foundation established in this dissertation and aim to provide even more valuable tools and insights for software developers and the broader SE community.

Bibliography

[1] Graphql. https://docs.github.com/en/graphql, 02 2022. (Accessed on 02/10/2022).

[2] hadoop. https://issues.apache.org/jira/projects/HADOOP/issues, 02 2022. (Accessed on 02/10/2022).

[3] Jira. https://www.atlassian.com/software/jira, Feb 2022. (Accessed on 02/10/2022).

[4] kibana. https://github.com/elastic/kibana, 2022. (Accessed on 02/10/2022).

[5] spark. https://issues.apache.org/jira/projects/SPARK/issues, 2022. (Accessed on 02/10/2022).

[6] vscode repository on github. https://github.com/microsoft/vscode, 2022. (Accessed on 02/10/2022).

[7] vscodebot on github. https://github.com/apps/vscodebot, 2022. (Accessed on 02/10/2022).

[8] (2) "too many requests in 1 hour. try again later." really annoying when will this be fixed? : Chatgpt. https://www.reddit.com/r/ChatGPT/comments/zm9g0n/too_many_requests_in_1_hour_try_again_later/, 2023. (Accessed on 05/05/2023).

[9] acheong08/chatgpt: Reverse engineered chatgpt api. https://github.com/acheong08/ChatGPT, 2023. (Accessed on 05/05/2023).

[10] axios/axios: Promise based http client for the browser and node.js. https://github.com/axios/axios, 2023. (Accessed on 05/05/2023).

[11] Bugzilla. https://www.bugzilla.org/, 2023. (Accessed on 05/05/2023).

[12] Chatgpt release notes | openai help center. https://help.openai.com/en/articles/6825453-chatgpt-release-notes, 2023. (Accessed on 05/05/2023).

[13] Github. https://github.com/, 2023. (Accessed on 05/05/2023).

[14] [hadoop-17091] [jdk11] fix javadoc errors - asf jira. https://issues.apache.org/jira/browse/HADOOP-17091, 2023. (Accessed on 05/05/2023).

[15] Introducing chatgpt. https://openai.com/blog/chatgpt, 2023. (Accessed on 05/05/2023).

[16] Jira | issue & project tracking software | atlassian. https://www.atlassian.com/software/jira, 2023. (Accessed on 05/05/2023).

[17] ohmyzsh/ohmyzsh. https://github.com/ohmyzsh/ohmyzsh, 2023. (Accessed on 05/05/2023).

[18] Significant-gravitas/auto-gpt. https://github.com/Significant-Gravitas/Auto-GPT, 2023. (Accessed on 05/05/2023).

[19] [spark-31169] random forest in sparkml 2.3.3 vs 2.4.x - asf jira. https://issues.apache.org/jira/browse/SPARK-31169, 2023. (Accessed on 05/05/2023).

[20] [spark-33661] unable to load randomforestclassificationmodel trained in spark 2.x - asf jira. https://issues.apache.org/jira/browse/SPARK-33661, 2023. (Accessed on 05/05/2023).

[21] vuejs/vue: This is the repo for vue 2. for vue 3, go to https://github.com/vuejs/core. https://github.com/vuejs/vue, 2023. (Accessed on 05/05/2023).

[22] T. Ahmed, A. Bosu, A. Iqbal, and S. Rahimi. Senticr: A customized sentiment analysis tool for code review interactions. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 106–111. IEEE, 2017.

[23] A. Alipour, A. Hindle, and E. Stroulia. A contextual approach towards more accurate duplicate bug report detection. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 183–192. IEEE, 2013.

[24] M. Allahyari, S. Pouriyeh, M. Assefi, S. Safaei, E. D. Trippe, J. B. Gutierrez, and K. Kochut. Text summarization techniques: a brief survey. *arXiv preprint arXiv:1707.02268*, 2017.

[25] M. Amoui, N. Kaushik, A. Al-Dabbagh, L. Tahvildari, S. Li, and W. Liu. Search-based duplicate defect detection: an industrial experience. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 173–182. IEEE, 2013.

[26] J. Arokiam and J. S. Bradbury. Automatically predicting bug severity early in the development process. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: New Ideas and Emerging Results*, pages 17–20, 2020.

[27] Atlassian. Open source project license request | atlassian. https://www.atlassian.com/software/views/open-source-license-request, 2022. (Accessed on 02/10/2022).

[28] A. Baarah, A. Aloqaily, Z. Salah, M. Zamzeer, and M. Sallam. Machine learning approaches for predicting the severity level of software bug reports in closed source projects. *International Journal of Advanced Computer Science and Applications*, 10(10.14569), 2019.

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

[30] R. K. Bakshi, N. Kaur, R. Kaur, and G. Kaur. Opinion mining and sentiment analysis. In *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, pages 452–455. IEEE, 2016.

[31] Y. Bang, S. Cahyawijaya, N. Lee, W. Dai, D. Su, B. Wilie, H. Lovenia, Z. Ji, T. Yu, W. Chung, et al. A multitask, multilingual, multimodal evaluation of chatgpt on reasoning, hallucination, and interactivity. *arXiv preprint arXiv:2302.04023*, 2023.

[32] V. R. Basili. The role of controlled experiments in software engineering research. In *Empirical Software Engineering Issues. Critical Assessment and Future Directions*, pages 33–37. Springer, 2007.

[33] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim. Duplicate bug reports considered harmful... really? In *2008 IEEE International Conference on Software Maintenance*, pages 337–345. IEEE, 2008.

[34] A. Bhattacharjee, S. S. Nath, S. Zhou, D. Chakroborti, B. Roy, C. K. Roy, and K. Schneider. An exploratory study to find motives behind cross-platform forks from software heritage dataset. In *Proceedings of the 17th International Conference on Mining Software Repositories*, pages 11–15, 2020.

[35] E. Biswas, M. E. Karabulut, L. Pollock, and K. Vijay-Shanker. Achieving reliable sentiment analysis in the software engineering domain using bert. In *2020 IEEE International conference on software maintenance and evolution (ICSME)*, pages 162–173. IEEE, 2020.

[36] R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

[37] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[38] J. D. Brutlag, H. Hutchinson, and M. Stone. User preference and search engine latency. 2008.

[39] Bugzilla. Eclipse. https://bugs.eclipse.org/bugs/, 2022. (Accessed on 02/10/2022).

[40] Bugzilla. Mozilla. https://bugzilla.mozilla.org/home, 2022. (Accessed on 02/10/2022).

[41] I. Cachola, K. Lo, A. Cohan, and D. S. Weld. Tldr: Extreme summarization of scientific documents. *arXiv preprint arXiv:2004.15011*, 2020.

[42] F. Calefato, F. Lanubile, F. Maiorano, and N. Novielli. Sentiment polarity detection for software development. *Empirical Software Engineering*, 23(3):1352–1382, 2018.

[43] F. Calefato, F. Lanubile, N. Novielli, and L. Quaranta. Emtk-the emotion mining toolkit. In *2019 IEEE/ACM 4th International Workshop on Emotion Awareness in Software Engineering (SEmotion)*, pages 34–37. IEEE, 2019.

[44] A. Celikyilmaz, A. Bosselut, X. He, and Y. Choi. Deep communicating agents for abstractive summarization. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1662–1675, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

[45] N. Chen, S. C. Hoi, S. Li, and X. Xiao. Simapp: A framework for detecting similar mobile applications by online kernel learning. In *Proceedings of the eighth ACM international conference on web search and data mining*, pages 305–314, 2015.

[46] 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 *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, pages 385–397, 2020.

[47] Z. Chen, Y. Cao, X. Lu, Q. Mei, and X. Liu. Sentimoji: an emoji-powered learning approach for sentiment analysis in software engineering. In *Proceedings of the 2019 27th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering*, pages 841–852, 2019.

[48] Z. Chen, Y. Cao, H. Yao, X. Lu, X. Peng, H. Mei, and X. Liu. Emoji-powered sentiment and emotion detection from software developers' communication data. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 30(2):1–48, 2021.

[49] W.-L. Chiang, Z. Li, Z. Lin, Y. Sheng, Z. Wu, H. Zhang, L. Zheng, S. Zhuang, Y. Zhuang, J. E. Gonzalez, I. Stoica, and E. P. Xing. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality, March 2023.

[50] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

[51] H. Dai, Z. Liu, W. Liao, X. Huang, Z. Wu, L. Zhao, W. Liu, N. Liu, S. Li, D. Zhu, et al. Chataug: Leveraging chatgpt for text data augmentation. *arXiv preprint arXiv:2302.13007*, 2023.

[52] M. De Choudhury and S. Counts. Understanding affect in the workplace via social media. In *Proceedings of the 2013 conference on Computer supported cooperative work*, pages 303–316, 2013.

[53] X. Deng, V. Bashlovkina, F. Han, S. Baumgartner, and M. Bendersky. Llms to the moon? reddit market sentiment analysis with large language models. In *Companion Proceedings of the ACM Web Conference 2023*, pages 1014–1019, 2023.

[54] 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)*, pages 115–124. IEEE, 2017.

[55] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, pages 4171–4186, 2019.

[56] W. S. El-Kassas, C. R. Salama, A. A. Rafea, and H. K. Mohamed. Automatic text summarization: A comprehensive survey. *Expert Systems with Applications*, 165:113679, 2021.

[57] B. W. FAQ. How to mark a bug report as a duplicate? https://wiki.documentfoundation.org/QA/Bugzilla/FAQ#How_to_Mark_a_Bug_Report_as_a_Duplicate, 2022. (Accessed on 02/10/2022).

[58] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, et al. Codebert: A pre-trained model for programming and natural languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1536–1547, 2020.

[59] T.-y. Fu, W.-C. Lee, and Z. Lei. Hin2vec: Explore meta-paths in heterogeneous information networks for representation learning. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 1797–1806, 2017.

[60] W. Fu and T. Menzies. Easy over hard: A case study on deep learning. In *Proceedings of the 2017 11th joint meeting on foundations of software engineering*, pages 49–60, 2017.

[61] D. Gachechiladze, F. Lanubile, N. Novielli, and A. Serebrenik. Anger and its direction in collaborative software development. In *2017 IEEE/ACM 39th International Conference on Software Engineering: New Ideas and Emerging Technologies Results Track (ICSE-NIER)*, pages 11–14. IEEE, 2017.

[62] V. Garousi, M. Felderer, M. V. Mäntylä, and A. Rainer. *Benefitting from the Grey Literature in Software Engineering Research*, pages 385–413. Springer International Publishing, Cham, 2020.

[63] E. A. Gehan. A generalized wilcoxon test for comparing arbitrarily singly-censored samples. *Biometrika*, 52(1-2):203–224, 1965.

[64] GitHub. Marking issues or pull requests as a duplicate - github docs. https://docs.github.com/en/issues/tracking-your-work-with-issues/marking-issues-or-pull-requests-as-a-duplicate. (Accessed on 02/10/2022).

[65] M. Golzadeh, A. Decan, E. Constantinou, and T. Mens. Identifying bot activity in github pull request and issue comments. In *2021 IEEE/ACM Third International Workshop on Bots in Software Engineering (BotSE)*, pages 21–25. IEEE, 2021.

[66] G. Gousios. The ghtorrent dataset and tool suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 233–236, Piscataway, NJ, USA, 2013. IEEE Press.

[67] G. Gousios, M. Pinzger, and A. v. Deursen. An exploratory study of the pull-based software development model. In *Proceedings of the 36th international conference on software engineering*, pages 345–355, 2014.

[68] G. Gousios, M.-A. Storey, and A. Bacchelli. Work practices and challenges in pull-based development: the contributor's perspective. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pages 285–296. IEEE, 2016.

[69] G. Gousios, A. Zaidman, M.-A. Storey, and A. Van Deursen. Work practices and challenges in pull-based development: The integrator's perspective. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pages 358–368. IEEE, 2015.

[70] J. Gu, Z. Lu, H. Li, and V. O. Li. Incorporating copying mechanism in sequence-to-sequence learning. *arXiv preprint arXiv:1603.06393*, 2016.

[71] D. Guo, S. Ren, S. Lu, Z. Feng, D. Tang, S. Liu, L. Zhou, N. Duan, A. Svyatkovskiy, S. Fu, et al. Graphcodebert: Pre-training code representations with data flow. *arXiv preprint arXiv:2009.08366*, 2020.

[72] P. J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy. Characterizing and predicting which bugs get fixed: an empirical study of microsoft windows. In *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering-Volume 1*, pages 495–504, 2010.

[73] Y. Guo, W. Qiu, Y. Wang, and T. Cohen. Automated lay language summarization of biomedical scientific reviews. *arXiv preprint arXiv:2012.12573*, 2020.

[74] E. Guzman, D. Azócar, and Y. Li. Sentiment analysis of commit comments in github: an empirical study. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 352–355, 2014.

[75] J. He, L. Xu, M. Yan, X. Xia, and Y. Lei. Duplicate bug report detection using dual-channel convolutional neural networks. In *Proceedings of the 28th International Conference on Program Comprehension*, pages 117–127, 2020.

[76] M. M. Imran, Y. Jain, P. Chatterjee, and K. Damevski. Data augmentation for improving emotion recognition in software engineering communication. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, pages 1–13, 2022.

[77] M. R. Islam, M. K. Ahmmed, and M. F. Zibran. Marvalous: Machine learning based detection of emotions in the valence-arousal space in software engineering text. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, pages 1786–1793, 2019.

[78] M. R. Islam and M. F. Zibran. Towards understanding and exploiting developers' emotional variations in software engineering. In *2016 IEEE 14th International Conference on Software Engineering Research, Management and Applications (SERA)*, pages 185–192. IEEE, 2016.

[79] M. R. Islam and M. F. Zibran. Leveraging automated sentiment analysis in software engineering. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 203–214. IEEE, 2017.

[80] M. R. Islam and M. F. Zibran. A comparison of software engineering domain specific sentiment analysis tools. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 487–491. IEEE, 2018.

[81] M. R. Islam and M. F. Zibran. Deva: sensing emotions in the valence arousal space in software engineering text. In *Proceedings of the 33rd annual ACM symposium on applied computing*, pages 1536–1543, 2018.

[82] M. R. Islam and M. F. Zibran. Sentistrength-se: Exploiting domain specificity for improved sentiment analysis in software engineering text. *Journal of Systems and Software*, 145:125–146, 2018.

[83] 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)*, pages 52–61. IEEE, 2008.

[84] J. Jiang, Q. Wu, J. Cao, X. Xia, and L. Zhang. Recommending tags for pull requests in github. *Information and Software Technology*, 129:106394, 2021.

[85] R. Jongeling, S. Datta, and A. Serebrenik. Choosing your weapons: On sentiment analysis tools for software engineering research. In *2015 IEEE international conference on software maintenance and evolution (ICSME)*, pages 531–535. IEEE, 2015.

[86] R. Jongeling, P. Sarkar, S. Datta, and A. Serebrenik. On negative results when using sentiment analysis tools for software engineering research. *Empirical Software Engineering*, 22:2543–2584, 2017.

[87] S. D. Joshi and S. Chimalakonda. Rapidrelease-a dataset of projects and issues on github with rapid releases. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 587–591. IEEE, 2019.

[88] A. Kanade, P. Maniatis, G. Balakrishnan, and K. Shi. Learning and evaluating contextual embedding of source code. In *International Conference on Machine Learning*, pages 5110–5121. PMLR, 2020.

[89] M. Kim and E. Lee. Are datasets for information retrieval-based bug localization techniques trustworthy? *Empirical Software Engineering*, 26(3):1–66, 2021.

[90] P. S. Kochhar, X. Xia, D. Lo, and S. Li. Practitioners' expectations on automated fault localization. In *Proceedings of the 25th International Symposium on Software Testing and Analysis*, pages 165–176, 2016.

[91] B. Kucuk and E. Tuzun. Characterizing duplicate bugs: An empirical analysis. In *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 661–668. IEEE, 2021.

[92] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut. Albert: A lite bert for self-supervised learning of language representations. In *International Conference on Learning Representations*, 2019.

[93] A. Lazar, S. Ritchey, and B. Sharif. Generating duplicate bug datasets. In *Proceedings of the 11th working conference on mining software repositories*, pages 392–395, 2014.

[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. *arXiv preprint arXiv:1910.13461*, 2019.

[95] Q. Lhoest, A. Villanova del Moral, Y. Jernite, A. Thakur, P. von Platen, S. Patil, J. Chaumond, M. Drame, J. Plu, L. Tunstall, J. Davison, M. Šaško, G. Chhablani, B. Malik, S. Brandeis, T. Le Scao, V. Sanh, C. Xu, N. Patry, A. McMillan-Major, P. Schmid, S. Gugger, C. Delangue, T. Matussière, L. Debut, S. Bekman, P. Cistac, T. Goehringer, V. Mustar, F. Lagunas, A. Rush, and T. Wolf. Datasets: A community library for natural language processing. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 175–184, Online and Punta Cana, Dominican Republic, Nov. 2021. Association for Computational Linguistics.

[96] X. Li, H. Jiang, D. Liu, Z. Ren, and G. Li. Unsupervised deep bug report summarization. In *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*, pages 144–14411. IEEE, 2018.

[97] 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)*, pages 136–140. IEEE, 2015.

[98] B. Lin, N. Cassee, A. Serebrenik, G. Bavota, N. Novielli, and M. Lanza. Opinion mining for software development: a systematic literature review. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 31(3):1–41, 2022.

[99] B. Lin, F. Zampetti, G. Bavota, M. Di Penta, M. Lanza, and R. Oliveto. Sentiment analysis for software engineering: How far can we go? In *Proceedings of the 40th international conference on software engineering*, pages 94–104, 2018.

[100] C.-Y. Lin. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81, 2004.

[101] J. Lin, Y. Liu, Q. Zeng, M. Jiang, and J. Cleland-Huang. Traceability transformed: Generating more accurate links with pre-trained bert models. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 324–335. IEEE, 2021.

[102] B. Liu et al. Sentiment analysis and subjectivity. *Handbook of natural language processing*, 2(2010):627–666, 2010.

[103] K. Liu, G. Yang, X. Chen, and C. Yu. Sotitle: A transformer-based post title generation approach for stack overflow. *arXiv preprint arXiv:2202.09789*, 2022.

[104] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys*, 55(9):1–35, 2023.

[105] W. Liu, S. Wang, X. Chen, and H. Jiang. Predicting the severity of bug reports based on feature selection. *International Journal of Software Engineering and Knowledge Engineering*, 28(04):537–558, 2018.

[106] Y. Liu and M. Lapata. Text summarization with pretrained encoders. *arXiv preprint arXiv:1908.08345*, 2019.

[107] 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. *arXiv preprint arXiv:1907.11692*, 2019.

[108] Z. Liu, X. Xia, C. Treude, D. Lo, and S. Li. Automatic generation of pull request descriptions. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 176–188. IEEE, 2019.

[109] P. Loyola, K. Gajananan, and F. Satoh. Bug localization by learning to rank and represent bug inducing changes. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 657–665, 2018.

[110] Y. Lu, M. Bartolo, A. Moore, S. Riedel, and P. Stenetorp. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8086–8098, 2022.

[111] K. Mahowald, A. A. Ivanova, I. A. Blank, N. Kanwisher, J. B. Tenenbaum, and E. Fedorenko. Dissociating language and thought in large language models: a cognitive perspective. *arXiv preprint arXiv:2301.06627*, 2023.

[112] W. Medhat, A. Hassan, and H. Korashy. Sentiment analysis algorithms and applications: A survey. *Ain Shams engineering journal*, 5(4):1093–1113, 2014.

[113] T. Menzies, S. Majumder, N. Balaji, K. Brey, and W. Fu. 500+ times faster than deep learning:(a case study exploring faster methods for text mining stackoverflow). In *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*, pages 554–563. IEEE, 2018.

[114] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[115] S. Min, X. Lyu, A. Holtzman, M. Artetxe, M. Lewis, H. Hajishirzi, and L. Zettlemoyer. Rethinking the role of demonstrations: What makes in-context learning work? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 11048–11064, 2022.

[116] L. Montgomery, C. Lüders, and W. Maalej. An alternative issue tracking dataset of public jira repositories. In *2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR)*, pages 73–77. IEEE, 2022.

[117] L. Moreno, G. Bavota, M. Di Penta, R. Oliveto, A. Marcus, and G. Canfora. Automatic generation of release notes. In *Proceedings of the 22nd acm sigsoft international symposium on foundations of software engineering*, pages 484–495, 2014.

[118] A. Murgia, M. Ortu, P. Tourani, B. Adams, and S. Demeyer. An exploratory qualitative and quantitative analysis of emotions in issue report comments of open source systems. *Empirical Software Engineering*, 23:521–564, 2018.

[119] D. Q. Nguyen, T. Vu, and A. T. Nguyen. Bertweet: A pre-trained language model for english tweets. *arXiv preprint arXiv:2005.10200*, 2020.

[120] J. Nielsen. *Usability engineering*. Morgan Kaufmann, 1994.

[121] N. Novielli, F. Calefato, D. Dongiovanni, D. Girardi, and F. Lanubile. Can we use se-specific sentiment analysis tools in a cross-platform setting? In *Proceedings of the 17th International Conference on Mining Software Repositories*, pages 158–168, 2020.

[122] N. Novielli, F. Calefato, F. Lanubile, and A. Serebrenik. Assessment of off-the-shelf se-specific sentiment analysis tools: An extended replication study. *Empirical Software Engineering*, 26(4):77, 2021.

[123] N. Novielli, D. Girardi, and F. Lanubile. A benchmark study on sentiment analysis for software engineering research. In *Proceedings of the 15th International Conference on Mining Software Repositories*, pages 364–375, 2018.

[124] N. Novielli and A. Serebrenik. Sentiment and emotion in software engineering. *IEEE Software*, 36(5):6–23, 2019.

[125] M. Obaidi, L. Nagel, A. Specht, and J. Klünder. Sentiment analysis tools in software engineering: A systematic mapping study. *Information and Software Technology*, page 107018, 2022.

[126] M. Ortu, A. Murgia, G. Destefanis, P. Tourani, R. Tonelli, M. Marchesi, and B. Adams. The emotional side of software developers in jira. In *Proceedings of the 13th international conference on mining software repositories*, pages 480–483, 2016.

[127] R. Paulus, C. Xiong, and R. Socher. A deep reinforced model for abstractive summarization. *arXiv preprint arXiv:1705.04304*, 2017.

[128] M. Pennacchiotti and A.-M. Popescu. Democrats, republicans and starbucks afficionados: user classification in twitter. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 430–438, 2011.

[129] E. Perez, D. Kiela, and K. Cho. True few-shot learning with language models. *Advances in neural information processing systems*, 34:11054–11070, 2021.

[130] M. Pradel, V. Murali, R. Qian, M. Machalica, E. Meijer, and S. Chandra. Scaffle: Bug localization on millions of files. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2020, page 225–236, New York, NY, USA, 2020. Association for Computing Machinery.

[131] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[132] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019.

[133] M. S. Rakha, C.-P. Bezemer, and A. E. Hassan. Revisiting the performance of automated approaches for the retrieval of duplicate reports in issue tracking systems that perform just-in-time duplicate retrieval. *Empirical Software Engineering*, 23(5):2597–2621, 2018.

[134] M. S. Rakha, W. Shang, and A. E. Hassan. Studying the needed effort for identifying duplicate issues. *Empirical Software Engineering*, 21(5):1960–1989, 2016.

[135] S. J. Rennie, E. Marcheret, Y. Mroueh, J. Ross, and V. Goel. Self-critical sequence training for image captioning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7008–7024, 2017.

[136] M. S. Riazi, B. D. Rouani, and F. Koushanfar. Deep learning on private data. *IEEE Security & Privacy*, 17(6):54–63, 2019.

[137] S. Robertson, H. Zaragoza, and M. Taylor. Simple bm25 extension to multiple weighted fields. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 42–49, 2004.

[138] I. M. Rodrigues, D. Aloise, E. R. Fernandes, and M. Dagenais. A soft alignment model for bug deduplication. In *Proceedings of the 17th International Conference on Mining Software Repositories*, pages 43–53, 2020.

[139] J. Romano, J. D. Kromrey, J. Coraggio, and J. Skowronek. Appropriate statistics for ordinal level data: Should we really be using t-test and cohen'sd for evaluating group differences on the nsse and other surveys. In *annual meeting of the Florida Association of Institutional Research*, volume 13, 2006.

[140] S. Romano, D. Fucci, M. T. Baldassarre, D. Caivano, and G. Scanniello. An empirical assessment on affective reactions of novice developers when applying test-driven development. In *International Conference on Product-Focused Software Process Improvement*, pages 3–19. Springer, 2019.

[141] X. Rong. word2vec parameter learning explained. *CoRR*, abs/1411.2738, 2014.

[142] P. Runeson, M. Alexandersson, and O. Nyholm. Detection of duplicate defect reports using natural language processing. In *29th International Conference on Software Engineering (ICSE'07)*, pages 499–510. IEEE, 2007.

[143] M. Salganicoff. *Tolerating Concept and Sampling Shift in Lazy Learning Using Prediction Error Context Switching*, pages 133–155. Springer Netherlands, Dordrecht, 1997.

[144] V. Sanh, L. Debut, J. Chaumond, and T. Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[145] K. R. Scherer, T. Wranik, J. Sangsue, V. Tran, and U. Scherer. Emotions in everyday life: Probability of occurrence, risk factors, appraisal and reaction patterns. *Social Science Information*, 43(4):499–570, 2004.

[146] H. Schütze, C. D. Manning, and P. Raghavan. *Introduction to information retrieval*, volume 39. Cambridge University Press Cambridge, 2008.

[147] A. See, P. J. Liu, and C. D. Manning. Get to the point: Summarization with pointer-generator networks. *arXiv preprint arXiv:1704.04368*, 2017.

[148] N. Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

[149] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.

[150] J. Su, Y. Lu, S. Pan, A. Murtadha, B. Wen, and Y. Liu. Roformer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864*, 2021.

[151] Y. Su, Z. Xing, X. Peng, X. Xia, C. Wang, X. Xu, and L. Zhu. Reducing bug triaging confusion by learning from mistakes with a bug tossing knowledge graph. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 191–202. IEEE, 2021.

[152] 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)*, pages 253–262. IEEE, 2011.

[153] 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*, pages 45–54, 2010.

[154] J. Tabassum, M. Maddela, W. Xu, and A. Ritter. Code and named entity recognition in stackoverflow. *arXiv preprint arXiv:2005.01634*, 2020.

[155] R. Taori, I. Gulrajani, T. Zhang, Y. Dubois, X. Li, C. Guestrin, P. Liang, and T. B. Hashimoto. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023.

[156] M. Thelwall, K. Buckley, G. Paltoglou, D. Cai, and A. Kappas. Sentiment strength detection in short informal text. *Journal of the American society for information science and technology*, 61(12):2544–2558, 2010.

[157] P. Tourani, Y. Jiang, and B. Adams. Monitoring sentiment in open source mailing lists: exploratory study on the apache ecosystem. In *CASCON*, volume 14, pages 34–44, 2014.

[158] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[159] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[160] F. Tu, J. Zhu, Q. Zheng, and M. Zhou. Be careful of when: An empirical study on time-related misuse of issue tracking data. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2018, page 307–318, New York, NY, USA, 2018. Association for Computing Machinery.

[161] G. Uddin and F. Khomh. Automatic mining of opinions expressed about apis in stack overflow. *IEEE Transactions on Software Engineering*, 2019.

[162] C. van der Lee, A. Gatt, E. van Miltenburg, and E. Krahmer. Human evaluation of automatically generated text: Current trends and best practice guidelines. *Computer Speech & Language*, 67:101151, 2021.

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

[164] L. Villarroel, G. Bavota, B. Russo, R. Oliveto, and M. Di Penta. Release planning of mobile apps based on user reviews. In *Proceedings of the 38th International Conference on Software Engineering*, pages 14–24, 2016.

[165] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun. An approach to detecting duplicate bug reports using natural language and execution information. In *Proceedings of the 30th international conference on Software engineering*, pages 461–470, 2008.

[166] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller. How long will it take to fix this bug? In *Fourth International Workshop on Mining Software Repositories (MSR'07: ICSE Workshops 2007)*, pages 1–1. IEEE, 2007.

[167] E. W. Weisstein. Bonferroni correction. *https://mathworld.wolfram.com/*, 2004.

[168] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, Q. Lhoest, and A. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, Oct. 2020. Association for Computational Linguistics.

[169] X. Xia, D. Lo, M. Wen, E. Shihab, and B. Zhou. An empirical study of bug report field reassignment. In *2014 Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*, pages 174–183. IEEE, 2014.

[170] G. Xiao, X. Du, Y. Sui, and T. Yue. Hindbr: Heterogeneous information network based duplicate bug report prediction. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, pages 195–206. IEEE, 2020.

[171] Q. Xie, Z. Wen, J. Zhu, C. Gao, and Z. Zheng. Detecting duplicate bug reports with convolutional neural networks. In *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*, pages 416–425. IEEE, 2018.

[172] C. Xu, Q. Sun, K. Zheng, X. Geng, P. Zhao, J. Feng, C. Tao, and D. Jiang. Wizardlm: Empowering large language models to follow complex instructions. *arXiv preprint arXiv:2304.12244*, 2023.

[173] S. Xu, Y. Yao, F. Xu, T. Gu, H. Tong, and J. Lu. Commit message generation for source code changes. In *IJCAI*, 2019.

[174] J. Xuan, H. Jiang, Z. Ren, J. Yan, and Z. Luo. Automatic bug triage using semi-supervised text classification. In *Proceedings of the 22nd International Conference on Software Engineering & Knowledge Engineering (SEKE'2010), Redwood City, San Francisco Bay, CA, USA, July 1 - July 3, 2010*, pages 209–214. Knowledge Systems Institute Graduate School, 2010.

[175] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in neural information processing systems*, pages 5753–5763, 2019.

[176] Y. Yu, H. Wang, V. Filkov, P. Devanbu, and B. Vasilescu. Wait for it: Determinants of pull request evaluation latency on github. In *2015 IEEE/ACM 12th working conference on mining software repositories*, pages 367–371. IEEE, 2015.

[177] Y. Yu, H. Wang, G. Yin, and T. Wang. Reviewer recommendation for pull-requests in github: What can we learn from code review and bug assignment? *Information and Software Technology*, 74:204–218, 2016.

[178] W. Yuan, P. Liu, and G. Neubig. Can we automate scientific reviewing? *arXiv preprint arXiv:2102.00176*, 2021.

[179] H. Zaragoza, N. Craswell, M. J. Taylor, S. Saria, and S. E. Robertson. Microsoft cambridge at trec 13: Web and hard tracks. In *Trec*, volume 4, pages 1–1, 2004.

[180] Z. Zeng, Y. Zhang, H. Zhang, and L. Zhang. Deep just-in-time defect prediction: how far are we? In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 427–438, 2021.

[181] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin, et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

[182] T. Zhang, D. P. Chandrasekaran, F. Thung, and D. Lo. Benchmarking library recognition in tweets. In *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension*, pages 343–353, 2022.

[183] T. Zhang, D. Han, V. Vinayakarao, I. C. Irsan, B. Xu, F. Thung, D. Lo, and L. Jiang. Duplicate bug report detection: How far are we? *ACM Transactions on Software Engineering and Methodology*, 2022.

[184] T. Zhang, I. C. Irsan, F. Thung, D. Han, D. Lo, and L. Jiang. Automatic pull request title generation. In *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 71–81. IEEE, 2022.

[185] T. Zhang, I. C. Irsan, F. Thung, and D. Lo. Cupid: Leveraging chatgpt for more accurate duplicate bug report detection. *arXiv preprint arXiv:2308.10022*, 2023.

[186] T. Zhang, I. C. Irsan, F. Thung, and D. Lo. Revisiting sentiment analysis for software engineering in the era of large language models. *arXiv preprint arXiv:2310.11113*, 2023.

[187] T. Zhang, B. Xu, F. Thung, S. A. Haryono, D. Lo, and L. Jiang. Sentiment analysis for software engineering: How far can pre-trained transformer models go? In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 70–80. IEEE, 2020.

[188] W. Zhang, Y. Deng, B. Liu, S. J. Pan, and L. Bing. Sentiment analysis in the era of large language models: A reality check. *arXiv preprint arXiv:2305.15005*, 2023.

[189] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong, et al. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 2023.

[190] C. Zhou, Q. Li, C. Li, J. Yu, Y. Liu, G. Wang, K. Zhang, C. Ji, Q. Yan, L. He, et al. A comprehensive survey on pretrained foundation models: A history from bert to chatgpt. *arXiv preprint arXiv:2302.09419*, 2023.

[191] J. Zhou and H. Zhang. Learning to rank duplicate bug reports. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 852–861, 2012.

[192] J. Zhu, M. Zhou, and A. Mockus. Effectiveness of code contribution: From patch-based to pull-request-based tools. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 871–882, 2016.

[193] 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*, pages 19–27, 2015.

[194] I. Žliobaitė, M. Pechenizkiy, and J. Gama. *An Overview of Concept Drift Applications*, pages 91–114. Springer International Publishing, Cham, 2016.

[195] W. Zou, D. Lo, Z. Chen, X. Xia, Y. Feng, and B. Xu. How practitioners perceive automated bug report management techniques. *IEEE Transactions on Software Engineering*, 46(8):836–862, 2018.