Reinforcement Learning for GitHub Pull Request Predictions: Analyzing Development Dynamics

Thesis by
Rinkesh Joshi

A thesis submitted to the Faculty of Graduate and Postdoctoral Affairs in partial fulfillment of the requirements for the degree of Master of Applied Science: Electrical and Computer Engineering (Specialization in Data Science)

Carleton University
Ottawa, Ontario

©2023
Rinkesh Joshi
All rights reserved
ABSTRACT

In the rapidly changing software development field, the pull-based model, supported by tools like GitHub, plays a pivotal role in collaborations. Understanding factors influencing this model is crucial for process enhancement. This thesis employs two Reinforcement Learning (RL) formalizations to predict Pull Request (PR) outcomes. The first utilizes 72 PR characteristics (e.g., PR Size, Test Inclusion, Developers’ PR Experience, Programming Language), achieving a G-mean of 0.83. The second focuses solely on PR discussions, attaining a higher G-mean of 0.88. Both RL models outperform established techniques like Random Forest, XGBoost, and Naive Bayes. Additionally, the study explores PR factors and merge time through a survey of 22 developers, identifying key influencers such as PR Size and Reviewer Experience, while also revealing common PR review approaches. Concluding, the study outlines achievements, future directions, and establishes an RL-based PR outcome prediction framework, along with publishing specific datasets.
II

ACKNOWLEDGEMENTS

I would like to begin by expressing my deepest gratitude to my supervisor, Dr. Nafiseh Kahani. Her unwavering support, patience, and guidance from the very start to the end of this study have been invaluable. Her willingness to provide insights, share her wisdom, and invest her time has greatly shaped this research, and I am eternally thankful for her contributions.

To my family, who made this journey possible, my sincerest thanks. Being an international student, the opportunity presented by Carleton University has been a truly transformative experience. My father, Baldevbhai, my mother, Jayshriiben, and my brother, Jay, have provided constant encouragement and love. Your faith in me has been a driving force behind every success, and I am grateful beyond words. I wish to pay a small tribute to my late grandmothers, Jamnaben and Gangaben. Your memories and teachings continue to guide me, and this work is a testament to the values you instilled in me. To the extended Joshi family, thank you for your unending support and belief in my abilities.

Special thanks go to Carleton University’s Research Ethics Boards, whose assistance in conducting the online survey and approving the ethics application was instrumental to this research.

I am humbled by the contributions of each individual and institution mentioned here, and many others who have influenced this research. This thesis stands as a testament to the collective effort, support, and inspiration that have been generously provided to me. Thank you.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Abstract</strong></td>
<td>I</td>
</tr>
<tr>
<td><strong>Acknowledgements</strong></td>
<td>II</td>
</tr>
<tr>
<td><strong>List of Abbreviations</strong></td>
<td>VI</td>
</tr>
<tr>
<td><strong>List of Figures</strong></td>
<td>VIII</td>
</tr>
<tr>
<td><strong>List of Tables</strong></td>
<td>IX</td>
</tr>
<tr>
<td><strong>1 Introduction</strong></td>
<td>1</td>
</tr>
<tr>
<td>1.1 Motivation and Objective</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Key Contributions</td>
<td>3</td>
</tr>
<tr>
<td>1.3 Thesis Organization</td>
<td>4</td>
</tr>
<tr>
<td><strong>2 Literature Review</strong></td>
<td>6</td>
</tr>
<tr>
<td>2.1 Identify Factors that Influence PR Outcomes</td>
<td>6</td>
</tr>
<tr>
<td>2.2 Importance, Needs, and Evolution of Code Review in Pull-based Software Development Model</td>
<td>12</td>
</tr>
<tr>
<td>2.3 Use of Sentiment Analysis to study GitHub Pull Requests</td>
<td>14</td>
</tr>
<tr>
<td>2.4 Predictive Techniques for PR Outcome</td>
<td>15</td>
</tr>
<tr>
<td>2.5 Synthesis of Literature Review Findings</td>
<td>17</td>
</tr>
<tr>
<td><strong>3 Background</strong></td>
<td>20</td>
</tr>
<tr>
<td>3.1 Pull-based Software Development</td>
<td>20</td>
</tr>
<tr>
<td>3.1.1 Pull Requests on GitHub</td>
<td>21</td>
</tr>
<tr>
<td>3.2 Reinforcement Learning</td>
<td>22</td>
</tr>
<tr>
<td>3.2.1 RL Methods-Classification</td>
<td>24</td>
</tr>
<tr>
<td>3.3 Sentiment Analysis of PR Comments</td>
<td>26</td>
</tr>
<tr>
<td>3.4 Evaluation Metrics</td>
<td>29</td>
</tr>
<tr>
<td>3.5 Comparative Baselines</td>
<td>31</td>
</tr>
<tr>
<td><strong>4 Predicting GitHub PR Outcomes: RL Formalizations Utilizing PR Characteristics</strong></td>
<td>34</td>
</tr>
<tr>
<td>4.1 Dataset</td>
<td>34</td>
</tr>
<tr>
<td>4.1.1 Data sources</td>
<td>34</td>
</tr>
</tbody>
</table>
4.1.2 Data Pre-processing ........................................... 36

4.2 Methodology ......................................................... 40

4.2.1 PR Review Markov Decision Process ................. 40

4.2.2 Configurating RL Environment: State Space, Action Space, and Reward Functions for PR Outcome Prediction .... 41

4.2.3 Training Approach for RL-based Prediction Model .... 44

4.2.4 Training Parameters ........................................... 45

4.3 Comparative Analysis ........................................... 49

4.3.1 Experiments ...................................................... 49

5 RL Formalizations Utilizing PR Discussions .......................... 59

5.1 Dataset .............................................................. 60

5.1.1 Data sources .................................................... 60

5.1.2 Feature Extraction and Sentiment Analysis of PR comments 60

5.2 Methodology ......................................................... 64

5.2.1 PR Review Markov Process for PR Comments Dataset ... 65

5.2.2 Configurating RL Environment ......................... 66

5.2.3 Training Approach ........................................... 68

5.2.4 Training Parameters ........................................... 71

5.3 Comparative Analysis ........................................... 72

5.3.1 Experiments ...................................................... 73

5.3.2 Contrasting the Two RL Approaches .................... 77

6 Understanding GitHub PRs: An Empirical Study on Review Processes and Merge Times through Developer Survey .............. 79

6.1 Survey Design ....................................................... 79

6.2 Survey Response Analysis Framework .................. 82

6.3 Results .............................................................. 84

6.3.1 Demographic and Workload Analysis .................. 84

6.3.2 Developer Preferences for PR Merges and Review Interaction 87

6.3.3 Perceptions of Influential Factors for PR Outcome ........ 88

6.3.4 Perceptions of Influential Factors for PR Merge Time .... 90

6.3.5 Analysis and Thematic Interpretation of the PR Review Process .................................................... 93

6.3.6 Analysis and Thematic Interpretation of Factors to Ascert- tain Quality of PRs .............................................. 95

7 Conclusion and Future Work ...................................... 98

7.1 Conclusion .......................................................... 98

7.2 Future Research ..................................................... 100
V

7.3 Threats to Validity ................................................. 101

References .................................................................. 103

Appendices ................................................................. 110
### LIST OF ABBREVIATIONS

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A2C</td>
<td>Advantage Actor-Critic</td>
</tr>
<tr>
<td>ACC</td>
<td>Accuracy</td>
</tr>
<tr>
<td>ADASYN</td>
<td>Adaptive Synthetic Sampling</td>
</tr>
<tr>
<td>AUC-ROC</td>
<td>Area Under the Receiver Operating Characteristic Curve</td>
</tr>
<tr>
<td>CVCS</td>
<td>Centralized Version Control System</td>
</tr>
<tr>
<td>DDPG</td>
<td>Deep Deterministic Policy Gradient</td>
</tr>
<tr>
<td>DQN</td>
<td>Deep Q-Network</td>
</tr>
<tr>
<td>DVCS</td>
<td>Distributed Version Control Systems</td>
</tr>
<tr>
<td>FN</td>
<td>False Negatives</td>
</tr>
<tr>
<td>FP</td>
<td>False Positives</td>
</tr>
<tr>
<td>FPR</td>
<td>False Positive Rate</td>
</tr>
<tr>
<td>G-mean</td>
<td>Geometric Mean</td>
</tr>
<tr>
<td>GPU</td>
<td>Graphics Processing Unit</td>
</tr>
<tr>
<td>IR</td>
<td>Imbalance Ratio</td>
</tr>
<tr>
<td>MDP</td>
<td>Markov Decision Process</td>
</tr>
<tr>
<td>NaN</td>
<td>Not a Number</td>
</tr>
<tr>
<td>NLTK</td>
<td>Natural Language Text Processing</td>
</tr>
<tr>
<td>OSS</td>
<td>Open Source Software</td>
</tr>
<tr>
<td>PPO</td>
<td>Proximal Policy Optimization</td>
</tr>
<tr>
<td>PR</td>
<td>Pull Request</td>
</tr>
<tr>
<td>RL</td>
<td>Reinforcement Learning</td>
</tr>
<tr>
<td>ROS</td>
<td>Random Over-Sampling</td>
</tr>
<tr>
<td>RUS</td>
<td>Random Under-Sampling</td>
</tr>
<tr>
<td>SAC</td>
<td>Soft Actor-Critic</td>
</tr>
<tr>
<td>SMOTE</td>
<td>Synthetic Minority Over-Sampling</td>
</tr>
<tr>
<td>SVM</td>
<td>Support Vector Machine</td>
</tr>
<tr>
<td>TD3</td>
<td>Twin Delayed Deep Deterministic Policy</td>
</tr>
<tr>
<td>TN</td>
<td>True Negatives</td>
</tr>
<tr>
<td>TP</td>
<td>True Positives</td>
</tr>
<tr>
<td>TPR</td>
<td>True Positive Rate</td>
</tr>
<tr>
<td>VADER</td>
<td>Valence Aware Dictionary and Sentiment Reasoner</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>VCS</td>
<td>Version Control System</td>
</tr>
<tr>
<td>WEKA</td>
<td>Waikato Environment for Knowledge Analysis</td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Conceptual Framework of the Thesis</td>
<td>4</td>
</tr>
<tr>
<td>6.1</td>
<td>Scatter Plot of Participants vs. Mean Monthly PR Workload with Average and Outlier Reference Lines</td>
<td>86</td>
</tr>
<tr>
<td>6.2</td>
<td>Comparative Scatter Plot of Experience Level in Current Role and Experience in Distributed Projects against Mean Monthly PRs (Workload), Categorized by Role - Excluding the Outlier</td>
<td>87</td>
</tr>
<tr>
<td>6.3</td>
<td>Bar Chart of Average Mean Monthly PR Workload by Experience Bins (2-Year Intervals) in Current Role, Excluding Outlier</td>
<td>88</td>
</tr>
<tr>
<td>6.4</td>
<td>Predominant PR Merge Types Selected by Participants</td>
<td>89</td>
</tr>
<tr>
<td>6.5</td>
<td>Participants’ Preferred Locations for Providing Comments During the PR Review Process (No Responses for ‘In Individual Commits’ Option)</td>
<td>89</td>
</tr>
<tr>
<td>6.6</td>
<td>PR Outcome Factors: Effective Response Analysis</td>
<td>90</td>
</tr>
<tr>
<td>6.7</td>
<td>Distribution of Participant Responses on Influential Factors to PR Outcome: A Contrast Across Strongly Agree, Agree, Disagree, and Strongly Disagree Scales</td>
<td>91</td>
</tr>
<tr>
<td>6.8</td>
<td>PR Merge Time Factors: Effective Response Analysis</td>
<td>92</td>
</tr>
<tr>
<td>6.9</td>
<td>Distribution of Participant Responses on Influential Factors to PR Merge Time: A Contrast Across Strongly Agree, Agree, Disagree, and Strongly Disagree Scales</td>
<td>92</td>
</tr>
<tr>
<td>B.1</td>
<td>Visualizing the Phi Correlation Coefficient: Correlation Between Dichotomous Variables in Chapter 4</td>
<td>115</td>
</tr>
<tr>
<td>B.2</td>
<td>Visualizing the Point-Biserial Correlation Coefficient: Correlation Between Continuous and Dichotomous Variables in Chapter 4</td>
<td>116</td>
</tr>
<tr>
<td>B.3</td>
<td>Visualizing the Point-Biserial Correlation Coefficient: Correlation Between Continuous and Dichotomous Variables in Chapter 5</td>
<td>117</td>
</tr>
<tr>
<td>B.4</td>
<td>Visualizing the Phi Correlation Coefficient: Correlation Between Dichotomous Variables in Chapter 5</td>
<td>118</td>
</tr>
<tr>
<td>B.5</td>
<td>Factor’s Influence on PR Outcome: Survey Response Distribution and Perceived Importance</td>
<td>119</td>
</tr>
<tr>
<td>B.6</td>
<td>Factor’s Influence on PR Merge Time: Survey Response Distribution and Perceived Importance</td>
<td>120</td>
</tr>
</tbody>
</table>
## LIST OF TABLES

2.1 Summary of Key Influential Factors: Descriptions, Identification Sources, and Statistical Significance .................................. 19

3.1 RL Algorithms in StableBaselines3 and Available Action Spaces  ................................................................................. 26

4.1 A Comprehensive List of All the Factors in the Processed Dataset .............................................................................. 36

4.2 Details and Distribution of PR Dataset: An 80/20 Split Analysis .................................................................................. 39

4.3 Training Parameter and Values of the learn() Method ................................................................................................. 47

4.4 Parameters and Values of predict() Method .................................................................................................................. 47

4.5 Performance of A2C Algorithm for PR Outcome Prediction with Different Values of Trade-Off Factor (k) ........................................................................................................... 50

4.6 Performance of PPO Algorithm for PR Outcome Prediction with Different Values of Trade-Off Factor (k) ........................................................................................................................................ 51

4.7 Performance of DQN Algorithm for PR Outcome Prediction with Different Values of Trade-Off Factor (k) ........................................................................................................................................ 51

4.8 Performance Evaluation of DQN Algorithm with Trade-Off Factor (2*IR) for Various Total Timesteps in an 80/20 Data Split Scenario .................................................................................. 52

4.9 Performance Evaluation of DQN Algorithm with Trade-Off Factor (2*IR) for Various Total Timesteps in a 50/50 Data Split Scenario .................................................................................. 53

4.10 Performance Evaluation of DQN Algorithm with Trade-Off Factor (2*IR) for Various Total Timesteps in a 20/80 Data Split Scenario .................................................................................. 53

4.11 Comparative Performance of Predictive Models with 80/20 Data Split and Resampling Techniques ........................................................................................................................................... 55

4.12 Comparison of Impact of Training Sample Size on Predictive Model Performance ........................................................................................................................................... 56

4.13 Evaluation of Predictive Models for Special Test Cases: Highlighting the Importance of G-mean Score ........................................................................................................................................... 57

5.1 A Comprehensive List of Features in the Curated PR Comments Dataset ........................................................................... 61

5.2 VADER Performance Analysis: Classification Report and G-mean Score ........................................................................... 63

5.3 Details and Distribution of PR Comments Dataset: An 80/20 Split Analysis ........................................................................... 64
| 5.4 | Performance of A2C Algorithm for Outcome Prediction with PR Comments Dataset | 74 |
| 5.5 | Performance of PPO Algorithm for Outcome Prediction with PR Comments Dataset | 74 |
| 5.6 | Performance of DQN Algorithm for Outcome Prediction with PR Comments Dataset | 74 |
| 5.7 | Comprehensive Comparative Performance Analysis of All Predictive Models | 76 |
| 6.1 | Survey Questions and Response Types for GitHub PR Review Process Study | 80 |
| 6.2 | Analysis of Steps Taken in PR Reviews: Initial Codes, Axial Codes, and Final Themes (Based on 22 Responses) | 95 |
| 6.3 | Analysis of Factors Used to Examine PR Quality: Initial Codes, Axial Codes, and Final Themes (Based on 22 Responses) | 97 |
| A.4 | Summary of Literature Review: Authors, Years, Datasets, and Methodological Remarks (Part-1) | 113 |
| A.5 | Summary of Literature Review: Authors, Years, Datasets, and Methodological Remarks (Part-2) | 114 |
Chapter 1

Introduction

1.1 Motivation and Objective

The emergence of Distributed Version Control Systems (DVCS) has led to the increase in adoption of the pull-based software development model. A growing number of developers are using pull-based development for distributed projects such as those hosted on GitHub, for both closed source as well as open source projects. This approach is particularly vital for Open Source Software (OSS) development, where collaboration and community contributions are key [1].

GitHub, a hosting platform for Git DVCS repositories, allows developers to fork code repositories and merge local branches to the master branch through Pull Request (PR). Hence, in a pull-based software development model, the developers (contributors) modify an isolated copy of the project’s repository, and then submit a PR to the repository owners. The owners (integrators) then evaluate the suggested changes in PR through an interactive review process and they either approve or reject the proposed merge into the main codebase of the project [2].

During the review process, other individuals from the development community can also actively engage in discussions, offering insights into the proposed changes. Furthermore, the process benefits from the presence of DevOps tools, which focus on the collaboration and communication of both software developers and IT professionals to automate the process of software delivery and infrastructure changes, specifically designed to streamline and enhance the development workflow. These tools automatically assess code compatibility, adherence to coding standards, and potential errors. The results of these automated checks are then shared with both the contributors and integrators to help with the PR review process [3]. This process promotes a decentralized and inclusive environment, fostering innovation and enabling more robust code development within the OSS community. Approximately 94 million developers actively utilize GitHub, with 52 million new open source projects initiated and over 413 million contributions made in 2022 alone [1].

The code velocity, or the rate at which code changes are examined and integrated, serves as a critical measure in software development. This pace not only enhances an engineer’s satisfaction with their work but also influences the overall efficiency of the code review process [4, 5]. Timely completion of PR reviews enables developers to better manage their workflow and preemptively mitigate potential project-wide delays. Conversely, the ramifications of prolonged PRs are multifaceted. These long-lived PRs can obstruct team communication by concealing a developer’s work, slow down the delivery of value to customers, and create complex merge conflicts if they remain diverged from the base repository for ex-
tended periods [6]. Recognizing these challenges, this research specifically aims to contribute by developing predictive models for the outcome of PR reviews, thereby enhancing the effectiveness of the code review process.

Numerous studies and authors have focused on identifying and studying the influential factors for PR outcome and merge time [2, 4, 8, 7, 11, 12, 13]. Georgios Gousios et al. [14, 15] as well as Caitlin Sadowski et al. [16] have comprehensively studied the review process from both the contributor’s and integrator’s perspectives. Erik van der Veen et al. [17], Motahareh Bahrami Zanjani et al. [18], and Guoliang Zhao et al. [19] presented PR prioritization tools, and effort estimation models were proposed by Chandra Maddila et al. [6, 20]. Studies on the comprehensive examination of the code review process were conducted by Jason Tsay et al. [21], Jiaxin Zhu et al. [22], Oleksii Kononenko et al. [23, 24], Zhi-Xing Li [24], and Yue Yu et al. [25]. Outside of the predictive approaches for PR review outcomes proposed by Tapajit Dey et al. [26], Jing Jiang et al. [27], and Çağdaş Evren Gerde et al. [28], there has been very limited research focusing on this aspect.

Inspired by the existing studies, our work focuses on accurately predicting the outcome of PR reviews. However, it differs from the existing studies in that it uses Reinforcement Learning (RL) and comprehensive datasets with a large sample size to make accurate predictions. We also compare the performance of existing machine learning techniques presented in existing studies with our RL formalizations. Further, our study revolves around a qualitative analysis of the available literature pertaining to the examination and identification of factors that are known to influence the decision-making process and merge time of PRs. This is achieved through a survey involving 22 developers, aiming to contribute to the existing body of research on GitHub PRs. By gathering contemporary, real-world responses, we capture updated insights and modern perspectives, thereby enhancing the relevance and applicability of our research findings.

In recent years, RL has been successfully applied to a range of fields such as computer gaming, recommendation and prioritization systems, and real-time ads bidding [29]. Unlike existing studies, which often rely on smaller sample sizes and a limited feature set for PR outcome prediction, we leverage the strengths of RL for this task. We model PR review outcome prediction as a sequential decision-making problem, capturing the interactions between the PR environment and a decision-making agent through RL. The interactive nature of RL offers a promising avenue for crafting predictive models that can handle large datasets and extensive feature sets. This approach has the potential to capture the nuanced PR review process that developers commonly follow, as evidenced by our developer survey. We proceeded to formalize and evaluate different RL configurations based on the processed datasets. We employed two distinct datasets to test two separate RL formalizations, with the aim of studying predictions of PR review outcomes from different perspectives. The first dataset encompasses a diverse range of factors associated with PRs on GitHub, including some that were identified as influential factors for review outcomes during the exhaustive literature review phase. This dataset provides a comprehensive view of PR characteristics. In contrast, the second dataset focuses solely on the PR discussions, specifically the comments contributed by both contributors and integrators on individual PRs on GitHub. This dataset allows us to explore the predictive performance of
the RL agent based on the collaborative discussions surrounding the PRs. The juxtaposition of these two approaches provides a rich insight into the complexity of PR review predictions. Together, they complement each other by addressing the PR review process from both a macro-level and micro-level.

In contrast to conventional machine learning methods, where input/output pairs guide actions, RL requires the agent to actively accumulate experience about states, actions, transitions, and rewards for optimal behavior. Rather than simply learning from a dataset, RL engages in learning through continuous interactions. As highlighted in Chapter 5, there were instances during this research where traditional supervised predictive models drastically underperformed. In those same cases, the RL-based models not only succeeded but also set new benchmarks, demonstrating the potential and robustness of RL in comparison to traditional techniques.

To make our study more comprehensive, we also conducted an exploratory survey involving software developers and software engineers as part of our study. This survey aimed to gather valuable insights and perspectives from professionals directly involved in the PR process, further enriching our understanding of the subject matter.

1.2 Key Contributions

This thesis makes the following contributions:

- An extensive review and analysis of the existing literature to summarize various factors that exhibit significance in the context of review outcome for new contributions in a pull-based software development model. It serves as a valuable resource for researchers and practitioners alike, enabling a deeper understanding of the subject matter and guiding future investigations and decision-making processes.

- We have curated a comprehensive dataset of GitHub projects, comprising of 66,281 PRs and containing 15 factors relevant to PR analysis. In addition, our dataset includes a total of 588,097 PR review comments. These comments provide insights into the interactions between contributors, reviewers, and integrators, reflecting the dynamics of the review process.

- Our study presents two highly effective predictive models for PR review outcomes, utilizing distinct RL configurations. These models leverage two extensive sets of PR features, allowing for comprehensive and accurate predictions. The results demonstrated that the first formalization reached a top G-mean score of 0.83, and the second, which emphasized PR discussions, surpassed this with a best case G-mean of 0.88. By employing the Deep Q-Network (DQN) algorithm and a uniquely crafted cost-sensitive reward function, the RL-based models consistently bested traditional classifiers in numerous test scenarios, highlighting their flexibility and accuracy.

- The performance of a range of existing predictive models is assessed and compared with the proposed RL approaches through comprehensive testing using diverse test datasets and extensive experiments.

2https://zenodo.org/record/8271704
A survey conducted with professional full-time developers, providing valuable insights into their perspectives on the assessment process of PRs, as well as the factors that influence the outcome of reviews and the merge time.

Figure 1.1: Conceptual Framework of the Thesis

1.3 Thesis Organization

Fig. 1.1 provides an overview of the study’s framework, starting with a comprehensive Literature Review, followed by data collection and preprocessing, an exploratory developer survey, and concluding with two RL formalizations for PR outcome prediction. The subsequent sections of the thesis are structured as follows. Chapter 2 presents a comprehensive literature review comprising of sub-sections dedicated to examining influential factors for review outcomes, state of pull-based development on GitHub, and analysis of sentiment in developer comments. Chapter 3 delves into the necessary background information and prerequisites, facilitating an understanding of RL concepts, the pull-based development model, PRs, and GitHub as a Version Control System (VCS). Chapter 4 and Chapter 5 outline the methodology employed in this study, presenting detailed sub-sections that cover data sources and pre-processing techniques utilized. Furthermore, these chapters delve into the different RL formalizations, including specifications such as the action space, observation space, and reward functions. In addition, they include sections focused on the experiments performed and an
extensive comparative analysis. Chapter 6 outlines the empirical study conducted through an exploratory survey targeting software developers and software engineers. Finally, in Chapter 7, a summary of the results is provided, accompanied by a thorough discussion of the findings and an examination of the potential threats to validity.
Chapter 2

Literature Review

In this section, we conduct a comprehensive and systematic literature review to identify factors—known as influential factors—that impact pull request outcomes on GitHub. The review covers studies published from 2013 to 2023 and spans four main areas: i) factors influencing PR outcomes, ii) the importance, needs, and evolution of the pull-based software development model, iii) the use of sentiment analysis in studying GitHub pull requests, and iv) predictive techniques for PR outcomes. To systematically gather relevant literature, we primarily utilized Google Scholar as our search engine.

For the section on influential factors for PR outcomes, we used carefully selected keywords including ‘pull request,’ ‘GitHub,’ ‘pull-based software development,’ ‘factors,’ ‘pull request decision,’ ‘distributed software development,’ ‘merged,’ and ‘rejected.’ For section ii) focusing on the importance, needs, and evolution of the pull-based model, our keyword search included terms like ‘pull-based software development,’ ‘distributed software development,’ ‘software engineering,’ ‘pull request,’ and ‘code review.’ In section iii), which explores the use of sentiment analysis to study GitHub PRs, we used keywords such as ‘GitHub,’ ‘pull request,’ ‘comments,’ ‘discussions,’ and ‘sentiment analysis.’ Lastly, for section iv) on predictive techniques, the keywords included ‘predictive model,’ ‘predictions,’ ‘machine learning,’ ‘reinforcement learning,’ ‘pull request,’ ‘code review,’ ‘pull-based software development,’ and ‘distributed software development.’

These sections encapsulate the initial phase of our study, serving as a foundation upon which our research is built. The studies were selected based on relevance, quality of methodology, and contribution to the field, thereby ensuring a robust review process.

2.1 Identify Factors that Influence PR Outcomes

We were able to identify a number of studies that focused on identifying the influential factors for pull request outcomes on various distributed version control platforms and tools such as GitHub 1, Geritt 2, Phabricator 3, and Mercurial 4.

Valentina Lenarduzzi et al., (2019) 10 conducted a study with the goal of understanding whether quality flaws such as code smells, anti-patterns, security vulnerabilities, and coding style violations in pull request code affect the acceptance of pull requests. The data involved the analysis of 28 open-source Java projects, examining 4.7 million code quality flaws in 36,344 pull requests. The methodology included extracting pull requests and code quality flaws using the

1https://github.com/
2https://www.gerritcodereview.com/
3https://www.phacility.com/phabricator/
4https://www.mercurial-scm.org/
GitHub REST API and PMD tool, performing statistical analysis with contingency matrices, the X2 test, logistic regression, and ensemble classifiers, and manually inspecting 10% of both accepted and non-accepted PRs. The conclusions of the study found that PMD issues did not influence pull request acceptance, as confirmed through manual inspection. Good aspects of the study include its comprehensive approach and combination of quantitative and qualitative methods, while potential weaknesses include a focus solely on Java-based projects, lack of investigation into the accuracy of the PMD tool, and possible bias from a high concentration of Apache Software Foundation projects in the data.

Marcelino Campos et al., (2016) [11] conducted a study with the goal of discovering and defining Technical Debt in pull requests that might lead to their rejection. The data for the study was drawn from 1722 pull requests from six open-source Java projects such as IntelliJ, Elastic Search, Iosched, Picasso, Retrofit, and Storm, after preprocessing from an initial pool of 7639 pull requests. The methodology involved manual categorization of pull requests into seven types of Technical Debts, namely design, documentation, test, build, project convention, performance, and security debt. The conclusions showed that 206 out of 679 rejected pull requests were due to Technical Debt, with design being the most common category. The positive aspect of the study is the detailed categorization of Technical Debts and public availability of the dataset, encouraging further research. However, the limitations include the lack of explanation for project selection, subjective classification by one author without clear criteria, and potential lack of generalizability of the findings.

Xunhui Zhang et al., (2021) [3], aimed to identify factors impacting pull request decisions on GitHub, employing a systematic literature review and subsequent data mining and analysis. The data consisted of 3,347,937 pull requests with 96 features from 11,230 GitHub projects, encompassing a range of team sizes, programming languages, and activities. The methodology involved manual review of titles and abstracts, selection of relevant papers reporting factors affecting pull request decisions, and analyzing correlations between factors using the Spearman correlation coefficient, Cramer’s V, and partial Eta-squared. A mixed-effect logistic regression model was created to experimentally elucidate the influence on pull request outcomes, revealing 46 variables that affected decisions. Specific models for different scenarios were also developed, identifying 5 to 10 factors as the best descriptors of outcomes, with the ‘same user’ factor being the most critical determinant. In conclusion, the study succeeded in highlighting essential factors affecting pull request decisions, benefiting from a large, diverse dataset. However, limitations include the single-platform focus on GitHub, raising questions about generalizability to other platforms, and lack of clarity about the consequences of manually identifying and excluding certain factors before applying the models.

Gunnar Kudrjavets et al. (2022) [4] examined the code review process to analyze delays and identify methods to accelerate the review and merging of code changes. Gathering data from 569,914 code reviews from Gerrit and Phabricator, they identified two areas of time delays in the review process: (i) between code submission and response, and (ii) between acceptance and merging. After pre-processing, their dataset was reduced to 350,043 code reviews. The study found that reducing idle waiting time could significantly decrease the time-to-
merge metric, and suggested automation to lessen the gap between acceptance and merging. However, they noted that automation might increase time-to-merge by placing more scrutiny on initial changes. Minor code modifications and submissions by experienced authors were more likely to be accepted without iterations. Despite insightful findings, the authors acknowledged the complexity of maximizing time-to-first-response and the limitation that the results may not apply to non-open-source projects.

In a detailed study, Georgios Goisios et al., (2014) [1] analyzed pull-based software development using 291 projects from the GHTorrent corpus and nearly 2 million pull requests across GitHub. They studied factors influencing pull request lifespan, merging, and rejection, focusing on selected Ruby, Python, Java, and Scala projects. The authors utilized projects with over 200 pull requests from February 2012 to August 2013, and their feature selection was grounded in literature analysis and developer interviews. After employing pair-wise correlation analysis, 15 significant features were isolated and examined using six classification algorithms, with Random Forest yielding the best results. A 10-fold random selection cross-validation approach was employed for model assessment. Key findings included the influence of actively maintained system parts, project size, and the number of altered files on the merge decision; developer track records, project size and test coverage, and openness to external contributions significantly impacted merge time. Despite insightful findings, only 14% of active projects utilized pull requests, leading the authors to recommend further research on team formations, unique code reviewing procedures, and developer motivations in a highly open environment.

The study by Jason Tsay et al., (2014) [12] aimed to understand how information in transparent open-source software environments like GitHub is used to evaluate contributions, specifically analyzing the technical and social factors influencing pull request acceptance. The authors employed a dataset of 659,501 pull requests across 12,482 projects collected from GitHub using specific sampling criteria and gathered additional information on issues and comments. They applied a multi-level mixed effects logistic regression model that included pull request-level, submitter-level, and repository-level measures to predict the likelihood of pull request acceptance. The conclusions revealed that both technical contribution practices and social connections between the submitter and project manager significantly influenced evaluation. Notably, contributions that included test cases, were from submitters with higher community standing, and from those with a higher status in the project were more likely to be accepted. In contrast, highly discussed pull requests were less likely to be accepted. The study provided a comprehensive analysis, although the complexity of the methodology may limit its application to a specific context.

Daricelio Moreira et al., (2015) [8] conducted a study to identify factors influencing the acceptance or rejection of pull requests in open-source projects hosted on GitHub. Utilizing a dataset of 61,592 pull requests from 72 projects, obtained from the 11th Working Conference on Mining Software Repositories (MSR) and using GHTorrent data corpus, the authors analyzed attributes like project identifier, programming language, developer type, commits per pull request, files added/edited/removed, analysis time, and final status of the pull request. The Apriori algorithm was employed in the WEKA platform to extract association
rules that might impact pull request outcomes. Their findings revealed that programming language, number of commits, files added, external developer contribution, and whether it was the developer’s first pull request significantly affected merging time and outcome. Moreover, they found that merging speed correlated with pull request approval. The study’s methodology allows for insightful analysis, although the limited scope of the dataset (focused only on projects hosted on GitHub) may question the generalizability of the results to other platforms, and further research could include the extraction of source code metrics for more comprehensive insights.

Jason Tsay et al., (2014) [21] conducted a study to understand how developers evaluate and discuss pull requests in GitHub’s open work environment. They focused on “highly discussed” pull requests, defined as those with comments one standard deviation above the mean, and interviewed 47 GitHub users. From a dataset of 659,501 pull requests across 12,482 projects, they randomly selected 20 pull requests, resulting in 423 comments from 115 developers for analysis. This was supplemented by semi-structured interviews. The authors used a grounded theory approach to analyze how contributions were evaluated, identifying categories of interaction between different types of participants and examining how core and peripheral developers engaged in discussion to resolve issues. They found that stakeholders, including third parties, attempted to influence outcomes, and that the submitter’s prior interaction level with the project affected the discussion around the contribution. The study shed light on the dynamics of collaborations in an open environment, particularly the complexities involved in evaluating and discussing code contributions.

Oleksii Kononenko et al. (2016) [5] conducted a qualitative survey study on code review practices within the Mozilla project, focusing on developer perceptions of code review quality. By examining a year’s worth of contributions recorded in the Bugzilla issue tracking system, they identified 403 experienced developers and received 88 responses. The study used grounded theory methodology and manual coding to analyze survey data. Findings indicated that most developers were engaged in both writing and reviewing patches. Developers conducted reviews in Bugzilla, despite other available tools, and believed factors such as experience, reviewer choice, patch size and quality, and bug severity influenced review time and decisions. Code review quality was perceived as dependent on clear, timely feedback from knowledgeable peers. Challenges were identified in both technical aspects like code familiarity and complexity, and personal aspects like time management. The study’s limitations include potential researcher bias and limited generalizability, as only Mozilla’s core developers were surveyed.

Chandra Maddila et al. (2023) [20] developed Nudge, an end-to-end service designed to accelerate overdue pull request completion by sending reminders to the author or reviewers. Using effort estimation and machine learning, specifically linear regression, they predicted the completion time of pull requests, followed by activity detection to filter out active pull requests, and actor identification to nudge the right individual. The methodology included 10-fold cross-validation and was tested in Microsoft’s environment on 147 repositories. Nudge succeeded in reducing pull request resolution time by 60% for 8,500 pull requests and was later scaled to 8,000 repositories with similar results. Furthermore, 73% of notifications were positively received by developers. The study’s conclusion emphasizes
Nudge’s innovative approach and effectiveness in reducing resolution times, and it also points out areas for future research such as considering dependencies among pull requests. However, the study’s limitation lies in its specific application within Microsoft, potentially affecting the generalizability of the results.

Olga Baysal et al., (2016) conducted a thorough investigation into the code review process of two major open-source projects, WebKit and Google Blink, aiming to identify both technical and non-technical factors that could impact code review response time and outcomes. The data collected included 17,459 bugs and 34,749 fixes for WebKit and 18,177 bugs and 37,280 patches for Blink. The methodology involved analyzing technical aspects such as patch size, priority, and component, as well as non-technical aspects like organization, developer experience, and reviewer activity. Both projects underwent substantial data pre-processing, and the Mann-Witney U (MWW) test was employed for statistical analysis. The conclusions revealed that, while patch size and code section significantly influenced the code review process, non-technical aspects were equally impactful. In addition to technical considerations, such as patch size and component, non-technical factors like organization and reviewer activity influenced review time. Limitations of the study include the focus on two specific projects, possibly affecting the generalizability of the results, and the unclear criterion for classifying factors as technical or non-technical. Furthermore, the study’s findings warrant further investigation into the effects of omitted factors, adding complexity to applying the results to other open-source projects.

Rahul Iyer et al., (2021) aimed to investigate the impact of developers’ personality traits on the GitHub pull request evaluation process. They extracted the big five personality traits (Openness, Conscientiousness, Extraversion, Agreeableness, and Neuroticism) from 16,935 developers’ digital footprints in 1,860 projects. Their data was drawn from two sources, including over 26,000 projects, and filtered down to those with significant pull request activity. Utilizing mixed effects logistic regression in R, they found that certain personality traits such as openness and conscientiousness positively influenced pull request approval, while traits like extraversion negatively affected it. The study concluded that although the effect of personality traits is significant, social features remained more influential on the likelihood of pull request acceptance. One unclear aspect was the process used to identify defunct projects, possibly affecting the data’s comprehensiveness, yet the study highlighted the importance of personality traits, comparable to technical factors.

Oleksii Kononenko et al., (2018) conducted two studies focused on pull request merges of Active Merchant, a Shopify Inc. product. The goal was to understand the nature of merges and to identify factors influencing merge time and outcomes. In the first study, they extracted 1,657 pull-requests from Active Merchant’s GitHub repository between January 1, 2012, and October 1, 2016, refining it to 1,475 pull requests after manual review and applying filters. Various data including pull-request details, size, comments, and author affiliation were considered. They employed a Multiple Linear Regression Model to study pull-request review time and a Logistic Regression Model to study the review decision, finding that factors such as pull-request size, discussion participants, author experience, and affiliation were influential. In the second study, a survey with developers contributing to Active Merchant was conducted, emphasizing that pull-request
quality was related to its description, complexity, and revertability, and review quality was linked to feedback, tests, and developer discussions. A significant limitation of these studies was their focus solely on a single project, potentially restricting the findings’ generalizability, and a lack of diversity in survey respondents.

Panthip Pooput et al., (2018) [9] aimed to identify impact factors associated with the rejection of pull requests on GitHub using association rules in data mining. They selected the Ansible project from the top ten most reviewed projects between January and December 2017, using the Octoverse report. The dataset consisted of approximately 5.9K contributors and 8,411 pull requests, with over 90% being rejected. They identified 50 factors related to pull requests, such as code quality metrics, which they divided into four categories: pull request changes count, complexity metrics, lines of code, and pull request size. Using GitHub’s official APIs and the Waikato Environment for Knowledge Analysis (WEKA) tool, they analyzed the data with the Apriori method. They discovered 15 factors, including code chunk, file changes count, community labels, etc., that had strong connections with rejection. The methodology was detailed, employing two datasets and using specific association rules with a confidence value higher than 0.95. The study concluded with a successful identification of influential factors. A limitation was the focus on a single project that utilized Python, suggesting a need for more research on various projects and programming languages hosted on different distributed version control systems.

Three following three studies [15], [14], and [30] utilized surveys to collect qualitative data in order to study the factors that influence the code review process, the challenges, the procedures, time and outcome of the pull requests. In the first two studies, the researchers performed two rounds of the online survey, the first of which was a pilot survey with a small group of contributors to clarify questions and highlight emergent themes for further investigation. They then submitted the survey in the second or final round, which was supplemented with questions addressing themes that developed after the first round. Both the surveys in the studies were split into three logical sections such as: (i) demographic information, (ii) multiple choice or Likert-scale questions, and (iii) open-ended questions.

Georgios Gousios et al., (2015) [15] conducted a study to understand the working patterns and challenges of integrators in the pull-based development paradigm. The goal was to explore how integrators used this model, how they decided on accepting contributions, evaluated quality, prioritized application, and faced challenges. They used a mixed-method approach, including a large-scale survey of 749 integrators and quantitative data from the GHTorrent dataset. The survey, conducted from April 14 to May 1, 2014, consisted of open-ended questions, Likert scale questions, and multiple-choice questions. Most respondents were project owners with significant industry experience (experience of at least 7 years). They applied the kmodes clustering technique for analysis and used manual coding for open-ended questions. Key findings include the use of pull-based paradigm for code reviews and feature debates, prioritization of quality and project fit for accepting contributions, evaluation criteria such as adherence to project style and test coverage, and various prioritization strategies like focusing on bug fixes. Challenges involved maintaining quality and social issues like motivating contributors. The study made a publicly accessible dataset available
but faced potential bias and generalizability issues due to the limited group of integrators surveyed.

Georgios Gousios et al., (2016) [14] aimed to uncover how contributors prepare for contributions and the challenges they encounter in open-source projects on GitHub that use pull-based development. Using a mixed-method strategy of an online survey and data from GHTorrent, the study surveyed 645 top contributors, with most having over 7 years of software development experience. The authors randomly selected contributors from projects receiving at least one pull request weekly in 2013 and analyzed the responses with manual coding. The results revealed a strong desire among contributors to stay informed about project status, but difficulties were noted in communication, especially regarding inadequate integrator response. The study concluded that conversations within pull requests often focused on low-level issues, with contributors preferring other methods for communication. The findings also highlighted further areas of research, such as task prioritization, anticipated merge time, untangling code changes, and the impact analysis of pull-requests.

Di Chen et al., in their 2019 study [30], sought to reduce the work involved with partial replication and augmentation of qualitative studies by examining the factors influencing the outcomes of GitHub pull requests. The study employed both qualitative and quantitative methods, using data from 170 pull requests from 142 GitHub projects, in addition to 20 PRs from a prior study. The authors conducted a systematic literature review, mapping qualitative insights into questions answerable by crowdsourced workers, and used Mechanical Turk (MTurk) to gather human assessments. They also implemented a decision tree classifier, based on quantitative and qualitative features, for predicting PR acceptance. The methodology comprised systematic analysis, crowdsourcing study, quality control through ‘gold’ queries, and comparative investigation with previous findings. The results indicated that the mixed-method approach achieved a considerably higher prediction accuracy (F1 score = 90% vs. 68%), validating the original study’s outcomes. However, the limited scope of the analysis, specifically the 170 pull requests used, might be viewed as a drawback, considering the vast amount of PRs in the source GHTorrent data corpus. Additionally, the utilization of more complex classification methods might have further enriched the study.

In summary, this section has reviewed studies that identify various factors influencing PR outcomes. These studies have contributed to the understanding of the variables that play a role in PR reviews, ranging from code quality to developer interactions. The identified factors serve as a foundation upon which further research, including this study, can build.

### 2.2 Importance, Needs, and Evolution of Code Review in Pull-based Software Development Model

Oleksii Kononenko et al., in their 2015 study [23], aimed to investigate the quality of code reviews within Mozilla’s three top modules: Core, Firefox, and Firefox for Android. They explored the relationships between the reviewers’ inspections and personal and social factors. The data included 44,595 commits extracted from Mozilla’s repository between January 1, 2013, and January 1, 2014, linked to 21,337 distinct bug IDs. Employing a data mining approach, the authors
carefully extracted commits, related them to bugs, applied filters, and identified bug-inducing modifications using the SZZ algorithm. Their conclusions revealed that 54% of Mozilla code reviews missed bugs. Factors such as developer engagement in bug-fix discussions, review experience, review loads, and the technical impact of the change were found to be key predictors of review quality. The study’s thorough examination of code review quality offers valuable insights into potential improvements in the review process. However, the complexity of the methodology and reliance on specific tools may pose challenges for adaptation in different contexts or projects.

Jiaxin Zhu et al. (2016) [22] conducted a study to compare the effectiveness of code contributions using patch-based tools, such as mailing lists and issue trackers, with pull-based systems on DVCSs like GitHub. They evaluated contribution effectiveness through three aspects: contribution effort (acceptance rate and ignored contributions ratio), time interval (time until the first response and resolution), and contribution activeness (contribution frequency). The study consisted of two sub-studies: one focused on a specific Rails project that migrated to GitHub, and the other compared eight patch-based projects with four pull-based projects. Results indicated that pull request systems had reduced review times and more contributions, but the acceptance or rejection rates did not consistently favor one system over the other. The study emphasizes the significance of understanding the role of tools in managing contributors but lacks detailed criteria for project selection, potentially affecting the generalizability of findings.

Caitlin Sadowski et al. (2018) [16] performed a case study on Google’s code review process to determine its necessity and observe changes over time. They utilized 12 interviews, a survey with 44 respondents, and review logs for 9 million changes, focusing on Google’s internally built code review tool, CRITIQUE. Through a mixed qualitative and quantitative approach, they gathered insights into the motivations for code review at Google, developers’ impressions, and existing methodologies. Key findings included the identification of code review’s primary purpose at Google: transitioning from a research codebase to a production codebase to enhance readability and maintainability. The study identified five core themes that Google developers expect from code review: education, maintaining norms, gatekeeping, accident prevention, and tracking history. Notably, 70% of changes were committed within 24 hours of initial review, indicating a light-weight review process at Google compared to other similar projects.

In their 2020 study, Klissiomara Dias et al. [31] set out to investigate seven factors associated with modularity, size, and timing of developers’ contributions, aiming to minimize merge conflicts that can occur when developers modify the same code artifacts. The study explored modularity in the context of application slices (related model, view, and controller files in MVC framework projects), size in terms of the number of developers, commits, changed files, and lines in contributions, and timing in terms of contribution duration and conclusion delays. They analyzed 73,504 merge scenarios from 100 Ruby and 25 Python projects hosted on GitHub, all based on popular MVC frameworks, namely Rails for Ruby and Django for Python. The methodology included collecting data on the seven factors, reproducing the merge operation, observing conflict occurrence, and employing regression models and collinearity tests to assess the factors’ effects. The authors discovered that merge conflicts were more likely when contributions
were not modular and were larger in size. They also found that longer development durations increased conflict likelihood. However, no evaluated factor showed predictive power for the number of merge conflicts or the number of files with conflicts. These findings can lead to effective recommendations for development teams and the creation of merge conflict prediction models. A potential limitation of this study is its specific focus on the MVC framework and the exclusive use of Ruby and Python projects, which could affect the generalizability of the results and render them highly contextual.

This section has highlighted the significance of code review in the pull-based software development model. The reviewed literature underscores the evolving needs and best practices in code review, emphasizing its role in ensuring software quality and facilitating collaboration among developers.

2.3 Use of Sentiment Analysis to study GitHub Pull Requests

In 2014, Emitza Guzman et al. [32] conducted a study using lexical sentiment analysis to examine emotions in commit comments of various open-source projects, probing the relationship with factors such as the programming language, time and day of the week of the commit, team distribution, and project approval. The researchers drew comments from 90 top-starred software projects on GitHub, focusing on 29 projects with more than 200 comments, analyzing a total of 60,425 commit comments. Using SentiStrength, a specialized lexical sentiment extraction tool, they found a tendency toward neutrality in emotion scores. Among the insights were more negative comments in Java projects, higher positive polarity in more distributed teams, and increased negative emotion in comments on Mondays. However, the study suffered from a limited sample size, a narrow focus on commit comments, and a lack of empirical justification for the performance of SentiStrength.

Similarly, Aman Kumar et al., in 2022 [33], delved into the sentiments of developer comments on GitHub to explore their effects on productivity, code quality, and satisfaction. The study encompassed three aspects: the day of the week when the comment was made; emotions throughout the project; and emotions across different programming languages. By selecting top Python repositories on GitHub and employing SentiStrength for sentiment analysis, they gathered comments from issues, commits, and pull requests. Their analysis of three repositories for Python, Java, C, C++, and .NET revealed more negative emotions in comments on Mondays, higher negative polarity in issue-related comments, and more positive comments in Java and Python compared to C and C++. Nevertheless, this study also lacked an evaluation section for SentiStrength, lacked empirical evidence to support its performance, and failed to clarify the sample size of the comments used, thereby limiting its reliability.

The study by Daniel Pletea et al., (2014) [34] sought to explore the presence and sentiments of security-related discussions on GitHub, specifically within the context of commits and pull requests. They carefully analyzed a dataset containing 60,658 commit comments and 54,892 pull request comments, chosen from the Mining Challenge Dataset 2014. By employing the Natural Language Text Processing (NLTK) tool and manually curating a list of security-related keywords,
they were able to identify and analyze comments associated with security discussions. Their findings revealed that approximately 10% of all discussions in their dataset were centered around security, and these conversations often harbored more negative emotions compared to other subjects. The authors further emphasized the importance of educating developers on security concerns and emphasized rigorous testing for vulnerabilities to enhance overall project atmosphere. Although the study offers valuable insights into the nature of security-related dialogues on GitHub, it suffers from significant drawbacks, including a lack of dedicated performance evaluation for the sentiment analysis tool used, a potentially biased selection of security-related keywords, and the mismatch between the training data (movie reviews) and the GitHub comments for the sentiment analysis tool. The study represents a critical exploration of a specialized topic but leaves room for improvement in methodological rigor and justification.

The study by Mohammad Masudur Rahman et al., (2017) had the goal of differentiating between useful and non-useful code review comments, and based on this differentiation, proposed a prediction model called RevHelper. Using 1,116 code review comments from four commercial systems on GitHub within an anonymized company, they found that 44.47% of the comments did not induce code changes and were considered non-useful. The methodology involved a detailed examination of textual characteristics and reviewer’s experience, focusing on eight independent variables including reading ease, stop word ratio, question ratio, code element ratio, and conceptual similarity, as well as experience-based factors. Their findings demonstrated significant differences between useful and non-useful comments, with the former sharing more vocabulary with the changed code and containing more relevant elements. Additionally, reviewers’ prior experience with the file being reviewed positively influenced the usefulness of comments. RevHelper, based on Random Forests, achieved a prediction accuracy of 66%, outperforming baselines like Naive Bayes and Logistic Regression. However, the study suffers from limitations, notably its focus on a specific company, which may restrict the generalizability of findings, and a manual annotation process that may introduce biases. Furthermore, the marginal improvement in accuracy by RevHelper over the baseline models raises questions about the system’s true effectiveness.

In this section, we’ve explored studies that apply sentiment analysis to GitHub PRs. These works demonstrate the potential of sentiment analysis as a tool for understanding the emotional dimensions and the social interactions involved in PR reviews.

2.4 Predictive Techniques for PR Outcome

During the comprehensive literature review, we encountered several studies that aimed to predict the outcome of PR reviews on GitHub effectively. Among these studies, we have identified the most relevant and closely related ones, which are discussed in detail below.

Tapajit Dey et al. (2020) introduced a predictive model designed to predict the acceptance of PRs on GitHub, aiming to address the challenges of handling numerous concurrent PR review processes faced by the integrators. They employed a Random Forest model and collected a comprehensive dataset com-
prising 50 PR-related factors from the GitHub repository of 4218 popular NPM packages. From these 50 factors, they further identified 14 predictors that proved sufficient for the prediction task. To achieve this, they utilized the \( \text{rfcv} \) function from the \texttt{randomForest} R package, which employs a nested cross-validation procedure to assess the prediction performance of models with sequentially reduced number of predictors. Their dataset included a total of 483,988 PRs, and after fine-tuning the Random Forest predictive model, they achieved an impressive AUC-ROC value of 0.95 when predicting PR acceptance. As a result, they concluded that PR integrators can utilize their model for highly accurate assessments of open PRs’ quality, while PR creators can benefit from understanding which characteristics of their PRs are desirable from the integrator’s perspective.

In our study, we refer to this approach as \( R\text{Forest}_{\text{PR}} \).

In a similar study conducted in 2020, Jiang Jing et al. \cite{27} introduced a predictive method for accepted PRs on GitHub. Their proposed approach, called CTCPPre, is based on XGBoost, a supervised predictive model that utilizes a decision tree boosting algorithm. For their investigation, they collected a dataset comprising of 28 GitHub projects, consisting of a total of 221,096 PRs, of which 154,851 were accepted PRs. Notably, they used an imbalanced dataset in their study to reflect real-world scenarios. The researchers focused on a diverse set of PR-related factors, including code features, text features of PR descriptions, developer characteristics, and project attributes, summing up to a total of 17 factors in their dataset. To evaluate the performance of their model, they compared it with state-of-the-art predictive models using their dataset. Ultimately, their proposed CTCPPre model outperformed other models, achieving an AUC-ROC score of 0.76 and an average F1-score of 0.88. In our study, we designate their proposed model as \( \text{CTCPPre} \) and aim to further compare its performance with our RL-based predictive model for PR review outcomes.

Finally, Cagdas Evren Gerede et al. \cite{28} conducted a comprehensive study exploring various machine learning techniques to predict whether code reviews (CRs) on the Geritt platform would undergo revisions before approval. Their research focused on the Android project hosted on Geritt and involved the collection of code review data for 11,633 CRs made between October 2008 and January 2012. Specifically, they aimed to predict whether a CR would have zero or more revisions. From the collected dataset, they identified 7 influential features, including the number of CRs previously submitted by the owner, the number of CRs previously merged by the owner, the branch name the CR was developed on, the number of reviewers, the number of reviewers that are approvers, and the number of reviewers that are verifiers. Remarkably, these factors are quite similar to the factors identified in our PR Review dataset. The authors employed several machine learning algorithms, including Support Vector Machine (SVM), AdaBoost, Bagging, and Random Forest, using the \texttt{WEKA} tool to train the predictive models. Among these, Random Forest yielded the best overall accuracy, precision, and recall scores.

The studies reviewed in this section focus on various predictive models for PR outcomes. From machine learning techniques to statistical models, these works lay the groundwork for future research, including the RL-based approaches proposed in this study.
2.5 Synthesis of Literature Review Findings

Table 2.1 compiles the key quantifiable factors identified in previous studies that have an impact on the PR review outcome. While many studies focus on more general themes or aspects to assess their influence on PR review, the factors in this table have been specifically chosen for their quantifiable nature. These factors were selected based on a thorough review of the original studies from which they were identified. We carefully examined these studies, their data sources, and the descriptions provided to ensure that each factor is quantifiable and has been utilized in quantitative research. These factors were chosen not only for their descriptions but also because they were used as features in quantitative analysis in the original studies, reinforcing their quantifiable nature and their potential impact on PR outcomes. The table also contains a brief description about the factors along with details on their relationship with the PR outcome variable.

The findings of the literature review are summarized below:

- In this exhaustive review, we exclusively concentrate on factors related to PR outcome without considering their effects on the merge time or the time required to approve the PR. This focus aligns with the main objective of our study, which is predicting PR review outcomes. In contrast, other studies that emphasize PR merge time tend to explore areas such as prioritization of reviewers or PRs, reviewer recommendations.

- A significant overlap was observed between studies that identified factors affecting PR outcome and those pinpointing influences on PR merge time. Consequently, several factors mentioned in this review as impactful for PR outcome are also recognized as influential for the PR merge time.

- As shown in Table A.4 and Table A.5 (in Appendix A, Additional Tables), during the literature review, we found a number of studies that utilized the GHTorrent data corpus [36] to collect the initial dataset for research on pull request outcomes, code review analysis, etc. The projects included in the GHTorrent data source are all hosted on GitHub only which might limit the number of platforms and neglect the potential effects of such exclusions from the study. However, given its wide-scale adoption, GHTorrent can be considered a robust source for studies focusing on the PR process on GitHub.

- A number of studies adopted a mixed-method strategy for data collection and data analysis. The authors frequently performed surveys and, in some cases, even interviews to confirm their findings in addition to the pull-request data gathered from publicly available repositories on various platforms.

- Furthermore, we observed that a recurrent limitation of the studies was a lack of generalizability of the results. The findings in almost all of the studies are valid only under specific contexts such as the version control platform being used, type of the programming languages found in the code-base, the code review practices and guidelines in place, type of the project (open-source vs closed source), contributor and integrator’s affiliations, and so on.
• Another common observation across the studies is the use of correlation analysis. In general, correlation between continuous and categorical variables was studied using Eta-squared, Cramer’s V for categorical variables, and Spearman’s coefficient for continuous variables. A few research combined correlation analysis with association rules.

• Finally, only a small number of papers have examined how the quality of the code affects the outcome of a pull request. While many studies have taken the lines of source code (LOC) into consideration, future research may utilize additional software metrics, such as cyclomatic complexity, coupling/depth of inheritance, number of classes, function count, lines of executable code, number of tests.

• Almost every research we came across concerning sentiment analysis of GitHub comments neglected to include an assessment to validate the choice and application of the sentiment analysis tool. Moreover, none of the tools were specifically designed or trained for GitHub comments, which often contain technical elements like code suggestions or corrections. The studies also failed to address how to handle the presence of informal language in comments, including acronyms like “LGMT” (looks good to me) or “lol” (signifying laughter), and emoticons such as “:).”

• In the course of our literature review, we identified CTCPPre and RForest_{PR} as the leading techniques for predicting PR outcomes. While these studies employ relatively smaller sample sizes and a limited feature set, our study distinguishes itself by utilizing two large, distinct datasets. One dataset comprises a comprehensive 72-feature set, and the other focuses solely on PR discussions. These datasets enable us to offer new perspectives that focus on the nuances of PR review. In drawing parallels between approaches proposed in CTCPPre, RForest_{PR}, and our own RL-based models, we further demonstrate that our proposed approaches are designed to mimic both single-stage and multi-stage PR review processes, thereby capturing the complexity of real-world PR reviews.

• CTCPPre employs XGBoost Classifier as its underlying technique; however, the hyperparameters used for model tuning and final implementation were not explicitly detailed. In contrast, Rforest_{PR} leverages Random Forest Classifiers, with the model fine-tuned for their specific dataset: the optimum value for the \textit{n}\textit{tree} parameter was found to be 500, and for \textit{mtry}, it was 6.

• Consequently, these two techniques and their underlying methods will be incorporated in the comparative analysis sections of our proposed RL techniques.
<table>
<thead>
<tr>
<th>Key Influential Factor</th>
<th>Description</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>11</th>
<th>13</th>
<th>15</th>
<th>23</th>
<th>30</th>
</tr>
</thead>
<tbody>
<tr>
<td>PR Size</td>
<td>sum of added and removed LOC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># of commenting devs</td>
<td># devs participating in discussion</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PR author affiliation</td>
<td>An organization that PR author affiliates with</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PR author experience</td>
<td># prior PRs submitted by PR author</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PR reviewer experience</td>
<td># prior PRs reviewed</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>technical debt</td>
<td>documentation, test, build, project convention, performance, and security debts</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>contrib/inteX</td>
<td>contributor/integrator personality traits (openness, conscientiousness, extraversion, agreeableness, neuroticism)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>same_user</td>
<td>same contributor and integrator? -yes/no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lifetime_minutes</td>
<td># minutes from PR creation to latest close time</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>prior_review_num</td>
<td># previous reviews in a project</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>has_comments</td>
<td>PR has comment? -yes/no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>core_number</td>
<td>core member? -yes/no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>language</td>
<td>programming language</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>commits_files_touched</td>
<td># total commits on files touched by the PR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lifetime_minutes</td>
<td># minutes from PR creation to latest close time</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>first_pr</td>
<td>first pull request? -yes/no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>code chunk</td>
<td>a section of code that can range in size from a single character to several sentences</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>files_changed</td>
<td># files added, removed, or changed</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>blank_line_count</td>
<td># code lines that are empty lines</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>multi_line_count</td>
<td># multi-lines</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>clone_instances</td>
<td># clone instances in the clone class</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>clone_classes</td>
<td># clone classes with at least one clone instance</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>clone_complexity</td>
<td>sum of McCabe complexity of the code fragment corresponding to the clone class</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Halstead</td>
<td>Halstead’s measurement reply on program execution. It indicates the quality of the software and are intended to permit assessing vocabulary, number of operands, etc.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>style conformance</td>
<td>refers to contributor’s conformance to project style</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>test coverage</td>
<td>refers to whether the changes in the PR are covered by existing or new tests</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>requester_suc_rate</td>
<td>% of the submitters PRs got merged previously</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

▼ Negative and statistically significant
▲ Positive and statistically significant
■ Statistically significant and positive (categorical data)
□ Statistically significant and negative (categorical data)
☒ Statistically significant but relation uncertain

Table 2.1: Summary of Key Influential Factors: Descriptions, Identification Sources, and Statistical Significance
Chapter 3

Background

In this chapter, we lay the foundational groundwork required to comprehend the complexities and nuances of our research. The chapter is organized into five key sections, each providing essential background information and prerequisites for understanding the study. The first section delves into the pull-based software development model, a prevalent approach in modern software engineering. The second section introduces the fundamentals of RL, the machine learning paradigm central to our predictive models. The third section focuses on the sentiment analysis of pull request comments, illuminating the social and emotional aspects that come into play during code reviews. The fourth section outlines the evaluation metrics used to assess the performance of predictive models, followed by the fifth section which discusses the comparative baselines against which our models are evaluated.

3.1 Pull-based Software Development

In modern software development practices, the frequent modification of code and related files is an integral part of the process, encompassing activities like adding new features, testing, development, and even the removal of existing features. This complex and iterative evolution often demands the implementation of multiple revisions before the software reaches its final form.

In such a context, one potential strategy is to replicate the original code before executing any alterations, preserving the option to revert to a specific previous version if needed. But as the complexity and size of the software project scales up, the manual handling of these various iterations becomes an exhausting and error-prone task [39]. This challenge leads to the necessity for a Version Control System (VCS), particularly in projects extending beyond several hundred lines of code, or when collaborative development involves multiple developers. A VCS is a specialized software tool designed to carefully track and manage the changes made to the codebase and associated files as they evolve over time.

VCSs can be subdivided into Centralized Version Control System (CVCS) and Distributed Version Control System (DVCS). CVCS revolves around a single, centralized repository that requires network connection for user access. Conversely, DVCS operates on a decentralized model, providing each user with a local repository and the ability to work offline, requiring network connection only for sharing repositories with others. This distinction gives DVCS the edge in terms of storing the entire file history on each user’s machine and synchronizing local changes back to a server for team collaboration [40]. In the following, we will review the features and functions of VCS [39, 40, 41]:

1. **Centralized Repository.** The VCS offers a consolidated repository where
all the versions of the code and related files are archived, ensuring consistent access and management.

2. **History Tracking.** It records the detailed history of alterations made by various developers in collaborative projects, promoting transparency and traceability.

3. **Branching Capability.** Within the repository, developers have the flexibility to create a new version or branch of the codebase for every modification they undertake. This branching mechanism allows individualized work on different aspects of the project such as feature development, bug fixes, or code refactoring without affecting the main codebase.

4. **Merging Functionality.** Upon completion, these branches can be seamlessly merged back into the main codebase. This not only integrates the changes but also preserves the overall integrity of the software system.

Among various VCS tools, Git stands as a widely adopted system, while GitHub acts as a hosting platform specifically designed for Git repositories. In conclusion, by providing robust features like a centralized repository, detailed change tracking, and flexible branching and merging capabilities, a DVCS like Git streamlines the development process, particularly in collaborative environments. The integration with platforms like GitHub further extends these capabilities, making the management of complex software projects more efficient and effective [41].

### 3.1.1 Pull Requests on GitHub

Distributed development is designed to foster collaboration in software projects by allowing potential contributors to submit changes to a project that’s overseen by a core team. Here’s how the process works, particularly focusing on the GitHub platform [1]:

1. **Cloning and Independent Work**
   - **Cloning.** Contributors begin by creating a clone of the repository, essentially making an independent copy that they can work on.
   - **Independent Changes.** Contributors then make changes in their cloned repositories, working separately from one another.
   - **Pull Requests.** When contributors are ready to submit their changes to the main repository, they create what is known as a PR. This specifies a local branch to be combined with a branch in the main repository.

2. **Inspection and Merging by Core Team**
   - **Inspection.** A core team member is tasked with reviewing the changes and deciding whether to pull them into the project’s master branch.
   - **Further Changes.** If the initial changes are not satisfactory, additional modifications may be requested. Contributors then update their local branches with new commits.
• **Discussion Features.** GitHub streamlines this process by enabling easy project cloning and automating the generation of PRs. Moreover, it allows contextual discussions through comments on PRs, in-line code comments, and individual commit comments.

3. **GitHub’s Pull Request Mechanism**

• **Branch Specification.** PRs on GitHub contain a branch (either local or in another repository) from which a core team member pulls commits. GitHub automatically identifies the commits to be merged.

• **Submission and Inspection.** By default, PRs are sent to the base repository for examination. They can be updated with new commits or closed, and GitHub will automatically update the displayed commits.

• **Final Merging.** Once inspected and found satisfactory, the PR can be merged. While anyone can participate in the inspection process, only a core team member can execute the merge.

4. **Methods of Merging on GitHub**

• **GitHub Merge.** GitHub checks if a PR can be automatically merged without conflicts. If so, it applies the commits and logs the merge event.

• **Git Merge.** Git utilities can be utilized for merging when automatic merging is not an option. There are three main methods:
  – **Branch Merging.** Merges the PR branch into the base repository, maintaining both history and authorship. However, GitHub may not detect this as a merge event.
  – **Cherry-Picking.** Specific commits are chosen and applied to the base branch, preserving authorship but changing the unique commit identifier, meaning exact history cannot be maintained.
  – **Commit Squashing.** Complementary to the above, several consecutive commits can be combined into one. In this case, the author of the combined commit differs from the person who applied it.

3.2 **Reinforcement Learning**

RL problem is primarily concerned with the process of learning how to make decisions by interacting with an environment to achieve a specific goal. This learning paradigm has two main components: an agent and an environment [42, 43]. The agent is the learner and a decision-maker in the system. It observes the current state of the environment, chooses an action to perform, and receives feedback in the form of a reward signal. Everything outside of the agent constitutes the environment. It’s the context in which the agent operates, and it responds to the agent’s actions by transitioning to new states and providing rewards. The interaction between the agent and the environment occurs in discrete steps, and the process at each step \( t \) can be described as follows [42]:
1. **State Observation.** The agent observes the current state $s_t$ from the state space $S$, where $s_t \in S$.

2. **Action Selection.** The agent selects an action $a_t$ from the action space $A$, based on its current policy.

3. **Environment Transition.** The chosen action $a_t$ changes the environment’s state to $s_{t+1}$.

4. **Reinforcement Signal.** The agent receives a reward signal $r_t$, calculated by the reward function $R(s_t, a_t)$, representing the value of the state transition.

5. **Policy Optimization.** The agent’s behavior, or policy $\pi(a_t|s_t)$, aims to choose actions that maximize the cumulative value of the reinforcement signal over time. It learns to do this through various algorithms, adapting through systematic trial and error.

6. **Discount factor.** $\gamma$ is a parameter that determines the agent’s preference for immediate versus future rewards, weighing long-term rewards more with higher values and immediate rewards with lower values; $\gamma \in (0, 1]$.

7. **Episode.** An episode is a sequence of transitions from an initial state to a terminal state, represented as $\{(s_1, a_1, r_1), (s_2, a_2, r_2), \ldots, (s_z, a_z, r_z)\}$, where $(s, a, r)$ is a tuple of state, action, and immediate reward, and $z$ is the terminal step.

The state and action spaces can be either discrete or continuous. The agent’s ultimate goal is to find an optimal policy $\pi$, mapping states to actions, to maximize some long-term measure of reinforcement. The state transition probabilities $P(s_{t+1}|s_t, a_t)$ are quantified by the transition function $P$.

**RL vs. Supervised Learning:** RL distinguishes itself from the more traditional supervised learning in several key aspects:

- **No Input/Output Pairs.** Unlike supervised learning, RL does not provide the agent with input/output pairs showing the best actions. Instead, the agent learns by exploring actions and observing subsequent states and immediate rewards.

- **Active Exploration.** The agent must actively gather experience about states, actions, transitions, and rewards to act optimally. It is not merely learning from a given dataset but learning through interactions.

- ** Concurrent Evaluation.** In RL, the system’s performance evaluation often happens simultaneously with learning, guided by a reward function. This dynamic, interactive nature makes RL applicable to complex, sequential decision-making problems.

In RL, when a problem exhibits the Markov property, meaning that state transitions are independent of prior states or agent actions, it can be represented as a [Markov Decision Process (MDP)](https://en.wikipedia.org/wiki/Markov_decision_process). An MDP, in a mathematical framework, is
characterized by a 5-tuple \((S, A, P, R, \gamma)\), where \(\gamma\) is the discount factor that balances the agent’s inclination for immediate versus future rewards [43].

An MDP provides an appropriate model for RL problems, particularly when dealing with delayed reinforcement, and offers a powerful way to formulate various RL tasks [42]. It enables the problem to be understood in terms of state transitions and rewards, which are essential for finding optimal decision-making policy.

When the complete system model, including state transition probabilities \(P\) and the reward function \(R(s_t, a_t)\), is known, dynamic programming methods can be employed. Techniques like value iteration, and policy iteration are used to calculate value functions for a given policy to find an optimal policy. In contrast, if the system model is not known or only partially known, a model-free approach is taken. In this case, the agent treats the environment as a black box, relying solely on interactions with it to deduce the optimal policy. Information regarding \(P\) and \(R(s_t, a_t)\) is not accessible to the agent beforehand, and therefore, RL algorithms are utilized to learn from the environment directly [42].

In the context of this study, the RL problem is formulated as an MDP using a model-free approach, signifying that the agent learns through direct interaction without knowledge of the underlying transition and reward functions. This formulation aligns well with the nature of many RL problems, emphasizing the adaptability and practicality of the MDP model, whether or not the system model is available [42, 43, 44].

3.2.1 RL Methods-Classification

RL methods are multifaceted and can be classified into different categories depending on the attributes being considered. Specifically, in this thesis, we explore classifications based on the type of optimization—value-based methods or policy-based methods—as well as the learning strategy, which includes on-policy and off-policy learning. The following classification categorizes RL methods into value-based, policy-based, and actor-critic methods.

1. Value-Based Methods: Value-based methods focus on estimating the optimal action-value function \(Q^*(s, a)\). For any given state-action pair \((s, a)\), the action value \(Q_\pi(s, a)\) represents the anticipated future reward following policy \(\pi(a|s)\) [44, 43].

   - Definition. The optimal action-value is computed as \(\max_{a_t \in A} Q_\pi(s_t, a_t)\), and the action with the highest value is selected.
   - Approach. These methods use a function approximator \(Q_\pi(s, a; \theta)\) with parameters \(\theta\) to represent the action-value function.
   - Examples. Algorithms like Q-learning use a Q-table and Bellman equation, while Deep Q-Networks (DQN) employ neural networks to estimate the action value.
   - Convergence. Generally efficient in sampling but limited in guaranteed convergence, often requiring careful hyperparameter tuning.

2. Policy-Based Methods: Policy-based methods directly work on optimizing the policy \(\pi(a|s; \theta)\) rather than estimating action values [42, 45].
- **Definition.** The policy is parameterized with $\theta$, and updates are performed using gradient ascent on the expected cumulative reward.

- **Approach.** Instead of estimating values, these methods explore the action space directly to find the optimal policy.

- **Examples.** REINFORCE is a prominent example that uses Monte Carlo estimates for gradient computation.

- **Convergence.** Stable but typically sample-inefficient, meaning they guarantee convergence but often at a slow rate.

3. **Actor-Critic Methods:** Actor-critic methods blend aspects of both value-based and policy-based methods, achieving a synergy between the two \[42\].

- **Composition.** Comprises of two components, the actor (policy) and the critic (value function).

- **Actor.** Follows the policy and selects actions, updating policy parameters based on the critic’s evaluations.

- **Critic.** Learns the value function, evaluating the actor’s actions, and guiding the actor’s updates.

- **Convergence.** These methods often exhibit desirable and faster convergence properties.

In summary, these three categories of RL methods provide different paths to the common goal of learning optimal decision-making strategies from interaction with the environment. Value-based methods leverage action value estimations, policy-based methods seek to optimize the policy directly, and actor-critic methods combine both approaches for enhanced learning efficiency. The choice among these methods depends on specific problem characteristics and requirements, such as sample efficiency, stability, and convergence speed.

The action space and observation space are fundamental concepts in RL, defining how an agent interacts with and perceives its environment, respectively. Both of these spaces can manifest in discrete or continuous forms, but not all RL algorithms can handle both types.

- **Action Space.** This outlines the set of possible actions that the agent can perform within its environment. While some algorithms support both discrete and continuous action spaces, others may have restrictions, limiting their applicability depending on the problem’s action space requirements.

- **Observation Space.** It specifies what information the agent can access about its environment. Unlike the action space, most RL algorithms do not impose constraints on the observation space, allowing for flexibility in the types of information that can be processed.

- **StableBaselines3 Framework.** In this research, our concentration is on the application of RL techniques using the open-source StableBaselines3 framework \[46\]. Building upon OpenAI Baselines, it comes with comprehensive documentation and support for advanced state-of-the-art RL algorithms, including Advantage Actor-Critic (A2C), DQN, Proximal Policy Optimization (PPO), Deep Deterministic Policy Gradient (DDPG), Soft Actor-Critic (SAC), Twin Delayed Deep Deterministic Policy (TD3).
• **Action and Observation Space in StableBaselines3.** The available action spaces for each RL algorithm offered by StableBaselines3 are detailed in Table 3.1. Utilizing OpenAI’s gym library for Python, this framework offers various types of spaces for both action and observation [47]:

  - **Dict Spaces.** Dictionary of spaces
  - **Discrete Spaces.** Space consisting of finite subset of integers
  - **Box Spaces.** Continuous spaces
  - **MultiDiscrete Spaces.** Represents cartesian product of arbitrary Discrete spaces
  - **MultiBinary Spaces.** An n-shape binary space

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Learning</th>
<th>On/Off</th>
<th>Action Space</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Box</td>
</tr>
<tr>
<td>A2C</td>
<td>Actor-Critic</td>
<td>On-policy</td>
<td>✓</td>
</tr>
<tr>
<td>DDPG</td>
<td>Policy based</td>
<td>Off-policy</td>
<td>✓</td>
</tr>
<tr>
<td>DQN</td>
<td>Value based</td>
<td>Off-policy</td>
<td></td>
</tr>
<tr>
<td>PPO</td>
<td>Actor-Critic</td>
<td>On-policy</td>
<td>✓</td>
</tr>
<tr>
<td>SAC</td>
<td>Actor-Critic</td>
<td>Off-policy</td>
<td>✓</td>
</tr>
<tr>
<td>TD3</td>
<td>Policy based</td>
<td>Off-policy</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 3.1: RL Algorithms in StableBaselines3 and Available Action Spaces

By understanding the nature of the action and observation spaces, researchers can select appropriate algorithms that align with the problem’s requirements. Based on the type of learning, RL methods can be further classified as following [29]:

**On-Policy Learning.** On-policy learning methods attempt to learn a policy that is directly influenced by the agent’s exploration strategy. The exploration policy, through which the agent experiments with its actions, is the same as the target policy it is trying to learn. Essentially, the actions chosen for exploration are based on the current best estimate of the optimal policy. Examples in StableBaselines3 include A2C, PPO, and SAC.

**Off-Policy Learning.** Off-policy learning methods involve learning a target policy that is independent of the exploration strategy. This means that the data used to update the policy can be generated by following a different policy altogether. Exploration during the learning phase is not necessarily based on the policy being learned, enabling more flexibility. Examples in StableBaselines3 include DQN, DDPG, and TD3.

On-policy and off-policy methods represent two distinct approaches to policy learning in RL. The former focuses on learning a policy that is aligned with the agent’s exploration, while the latter allows the learned policy to be independent of the exploration strategy.

### 3.3 Sentiment Analysis of PR Comments

To conduct sentiment analysis on the in-line code comments collected in our PR comments dataset, we employed Valence Aware Dictionary and Sentiment...
Re Reasoner (VADER), proposed by CJ Hutto et al. in 2014 [48], as our primary tool. VADER is a lexicon and rule-based model designed specifically for discerning general sentiment. It is particularly effective in understanding the sentiment from short, informal and conversational text, which makes it suitable for analyzing GitHub comments.

In Chapter 2 we discuss several studies that had similarly focused on the sentiment analysis of GitHub comments. Many of these studies employed SentiStrength, a lexicon-based tool that characterizes text as having either positive or negative emotional content. SentiStrength’s operation is primarily based on determining polarity, or discerning whether a comment reflects negative, positive, or neutral emotion.

At the outset, it is essential to note that the following discussion offers a brief overview of the various types of sentiment lexicons commonly employed in sentiment analysis methods. These lexicons serve as prerequisites for understanding the sentiment analysis tools used in this study.

Lexicon-based sentiment analysis methods rest on the foundations of a sentiment lexicon. This lexicon is essentially a list of lexical features, namely words, that are labeled as either positive or negative based on their semantic orientation. There are different types of sentiment lexicons. Some of these lexicons, known as polarity-based lexicons, classify words into one of two categories: positive or negative. This classification is based on the context-free semantic orientation of the words, creating a binary classification system [48].

Other lexicons, referred to as valence-based lexicons, associate words with a valence score that represents the intensity of sentiment. This approach allows for a more fine-grained assessment of sentiment, as it takes into account not just the binary positive/negative categorization, but also the strength or intensity of the sentiment expressed. This added layer of nuance can be beneficial for applications that require a more detailed understanding of sentiment within a text [48].

Finally, there are context-aware lexicons that delve further into lexical properties, such as part-of-speech, to offer a higher level of context awareness. This approach is used on top of a binary polarity-based method or the nuanced valence-based method, leading to more accurate and contextually sensitive sentiment analysis.

While both VADER and SentiStrength are lexicon-based tools and popular in the domain of sentiment analysis, we opted for VADER as our primary tool for reasons that we will elaborate upon. This decision was based on several considerations and an evaluation of the unique benefits that VADER offers in the context of our research.

- VADER is a simple yet effective lexicon and rule-based tool designed for computational sentiment analysis. It is particularly adept at processing social media style text, and its versatility allows it to generalize across multiple domains. One of the significant advantages of VADER is that it does not require any training data. Instead, it leverages a universally applicable, valence-based, human-curated gold standard sentiment lexicon.

---

1. [http://sentistrength.wlv.ac.uk/](http://sentistrength.wlv.ac.uk/)
2. [https://github.com/cjhutto/vaderSentiment](https://github.com/cjhutto/vaderSentiment)
Lexicon-based methods employ pre-defined dictionaries of words with assigned sentiment scores to calculate the overall sentiment of a text. On the other hand, rule-based approaches utilize manually crafted rules and heuristics to detect sentiment patterns within the text. VADER uniquely combines these two approaches, which allows it to conduct finely tuned and context-specific sentiment analysis. This hybrid methodology enhances its ability to accurately gauge the sentiment of a piece of text.

- VADER is a self-sufficient and domain-independent sentiment analysis tool that’s equipped to handle a wide array of linguistic nuances. It effectively manages negations such as “not good” and contractions like “isn’t good”, demonstrating a grasp of variations in tone and meaning. Moreover, VADER recognizes the impact of punctuation and word shapes in emphasizing sentiment intensity. For instance, it can differentiate the varying levels of excitement conveyed in “Yay, Another good day” versus “Yay! Another good day!” versus “YAY!! Another GOOD day.” (the understanding extends to the distinction between periods and exclamation points and the emphasis provided by capitalization).

VADER is also adept at accounting for degree modifiers that influence the intensity of sentiment. These include words that boost intensity such as “very”, and those that dampen intensity like “kind of”. Furthermore, VADER is attuned to informal language, recognizing slang words and acronyms like “lol”, “kinda”, and “sux”. Finally, the tool can comprehend emoticons, including utf-8 encoded emojis and traditional text emoticons like “:)” and “:+D”. This comprehensive ability to interpret a wide range of linguistic elements makes VADER an effective tool for sentiment analysis.

- Notably, VADER produces a positive, negative, and neutral score, each ranging from 0 to 1. These scores represent the proportions of text falling into each category, collectively summing up to 1. However, these proportions represent the basic categorization of individual lexical items, and they do not account for VADER’s rule-based enhancements such as sensitivity to word order, degree modifiers, amplification through word shape and punctuation, or negation polarity switches.

The compound score is a score adjusted by these rules, and it is normalized to range from -1 (representing extreme negativity) to +1 (indicating extreme positivity). In our PR Comments Dataset, we utilize the positive, negative, and neutral emotion scores along with this compound score. This combination provides a comprehensive view of the sentiment present in each comment.

- Thanks to VADER’s advanced capabilities discussed above, the raw comments do not necessitate extensive pre-processing before undergoing sentiment analysis. The minimal preparations involve excluding code suggestions, eliminating special characters, and removing URLs. Notably, elements like punctuation and stop words are retained in the text, reinforcing the tool’s ability to derive sentiment from these aspects.

- Lastly, VADER is a relatively contemporary tool in the field of sentiment
analysis. The authors compared its performance and effectiveness against both conventional benchmarks and advanced machine learning techniques for sentiment analysis.

On the other hand, SentiStrength is an older tool with relatively sparse documentation on its underlying operations. Unlike VADER, SentiStrength assigns scores ranging from -5 to +5, where positive numbers signify favorable attitudes and negative numbers denote unfavorable attitudes. While this does provide a measure of sentiment, it lacks the granularity of VADER’s system, which provides separate positive, negative, and neutral scores along with a compound score.

Therefore, based on these factors and our specific research requirements, we opted to use VADER for our sentiment analysis needs in analyzing GitHub PR comments.

### 3.4 Evaluation Metrics

In this study, we employ various evaluation metrics to measure and assess the accuracy of the prediction techniques. The evaluation metrics—Accuracy (ACC), F1-score, Geometric Mean (G-mean) score, and Area Under the Receiver Operating Characteristic Curve (AUC-ROC)—were selected based on their frequent use in the prior comparative works identified in our comprehensive literature review.

Traditionally, accuracy has been widely used as an empirical measure to evaluate predictive models. However, in the context of imbalanced datasets, accuracy alone may not provide an accurate representation of a model’s performance. This is because accuracy does not take into account the imbalanced distribution of classes and does not distinguish between correctly classified examples of different classes [49, 50].

To overcome this limitation, we utilize alternative evaluation metrics, such as the F1-score, G-mean score, and AUC-ROC. The F1-score considers both precision and recall and is more suitable for imbalanced datasets as it balances the trade-off between false positives and false negatives. The G-mean score is another metric that is sensitive to the imbalanced class distribution. Additionally, we use the AUC-ROC metric to assess the model’s ability to discriminate between positive and negative instances. AUC-ROC measures the model’s performance across different thresholds and is robust to imbalanced datasets [50].

- **Confusion Matrix.** A confusion matrix serves as a valuable tool to assess the performance of a classification algorithm. In the context of predictive analytics, the confusion matrix provides a comprehensive breakdown of the model’s predictions, distinguishing between True Positives (TP), False Negatives (FN), False Positives (FP), and True Negatives (TN).

  When the predictive model correctly identifies a PR review as accepted (class-1), it is considered a TP. Similarly, when the model accurately predicts a PR review as rejected (class-0), it is classified as a TN. On the other hand, if the model incorrectly predicts a PR review as accepted when it should be rejected, it becomes a false positive FP. Likewise, a false negative FN occurs when the model mistakenly predicts a rejection outcome for a PR review that should have been accepted.
• **Accuracy.** For a binary predictive model, accuracy can be calculated as the proportion of correctly classified instances (both true positives and true negatives) out of the total number of instances in the dataset. Mathematically, ACC can be defined as:

\[
ACC = \frac{TP + TN}{TP + TN + FP + FN}
\]

where: TP represents True Positives, TN or True Negatives, FP or False Positives, and FN or False Negatives

• **F1-score.** The F1 score considers both precision and recall, making it a valuable metric for evaluating the performance of binary predictive models, especially in imbalanced datasets [50]. It provides a balanced measure of the model’s ability to correctly predict both actions and takes into account both false positives and false negatives. For individual classes, the F1 score can be defined mathematically as follows, for the positive class (i.e., class 1, representing the majority PR acceptance action):

\[
F1_{positive} = \frac{2 \times \text{precision}_{positive} \times \text{recall}_{positive}}{\text{precision}_{positive} + \text{recall}_{positive}}
\]

where:

\[
\text{precision}_{positive} = \frac{TP}{TP + FP}
\]

\[
\text{recall}_{positive} = \frac{TP}{TP + FN}
\]

For the negative class (i.e., class 0, representing the minority PR rejection action):

\[
F1_{negative} = \frac{2 \times \text{precision}_{negative} \times \text{recall}_{negative}}{\text{precision}_{negative} + \text{recall}_{negative}}
\]

where:

\[
\text{precision}_{negative} = \frac{TN}{TN + FP}
\]

\[
\text{recall}_{negative} = \frac{TN}{TN + FN}
\]

• **G-mean score.** It is a performance metric commonly used for evaluating the performance of binary predictive models, especially in imbalanced datasets [50]. It takes into account both sensitivity (True Positive Rate) and specificity (True Negative Rate) of the model [3]. The G-mean score can be defined mathematically as follows:

\[
\text{G-mean} = \sqrt{\text{True Positive Rate} \times \text{True Negative Rate}}
\]

where:

\[
\text{True Positive Rate} = \frac{TP}{TP + FN}
\]

[^3]: [http://glemaitre.github.io/imbalanced-learn/generated/imblearn.metrics.geometric_mean_score.html](http://glemaitre.github.io/imbalanced-learn/generated/imblearn.metrics.geometric_mean_score.html)
True Negative Rate = \frac{TN}{TN + FP}

In the context of an RL-based model to predict PR review outcomes, the G-mean score measures the balance between correctly predicting the acceptance (positive class) and rejection (negative class) outcomes. It provides a single metric that considers the performance of the model on both classes, making it suitable for imbalanced datasets where the number of samples in each class is significantly different. A higher G-mean score indicates better overall performance of the model in terms of correctly predicting both classes.

- **AUC-ROC.** It is another widely used performance metric for binary predictive models. It measures the ability of the model to discriminate between positive and negative samples across various classification thresholds. The AUC-ROC can be mathematically defined as follows:

Let \( P(\text{positive}) \) represent the probability that a sample is classified as positive (i.e., accepted PR) by the model, and \( P(\text{negative}) \) represent the probability that a sample is classified as negative (i.e., rejected PR) by the model. The ROC curve is obtained by plotting the True Positive Rate (TPR) against the False Positive Rate (FPR) at various classification thresholds. The TPR is the same as the True Positive Rate defined earlier, while the FPR is calculated as follows:

\[
FPR = \frac{FP}{FP + TN}
\]

The AUC-ROC score is then computed as the area under the ROC curve. It quantifies the model’s ability to distinguish between positive and negative samples, with a higher AUC-ROC score indicating better prediction ability.

Accuracy ranges from 0 to 100, with higher values indicating better performance, although it may be misleading in imbalanced datasets. The Confusion Matrix quantifies correct and incorrect classifications; high diagonal values suggest better performance. F1-Score and G-Mean Score both range from 0 to 1 and are particularly useful in handling imbalanced classes, with higher values signifying better results. Lastly, the AUC-ROC also ranges from 0 to 1, with a score of 1 indicating perfect classification. Understanding these metrics, their range of values, and what extreme values mean provides a nuanced outlook through which to evaluate model performance.

By employing a combination of these evaluation metrics, we can gain a comprehensive understanding of the predictive model’s performance and its ability to handle imbalanced datasets effectively. This approach allows us to make informed comparisons and draw meaningful conclusions about the effectiveness of the prediction techniques in the context of PR outcome prediction.

### 3.5 Comparative Baselines

Consistent with the findings from our comprehensive literature review, we have identified three predictive methods that will serve as benchmarks for evaluating
the effectiveness of our proposed RL-based predictive approach. These methods will be compared using various evaluation metrics, including accuracy, F1-score, G-mean score, and AUC-ROC. Since our datasets exhibit class imbalance, our RL-based approach already incorporates a cost-sensitive reward function to address this issue. To ensure a fair comparison with the benchmark methods, we will also explore various sampling techniques aimed at handling imbalanced data. These sampling techniques can balance the class distribution, thereby providing a more equitable basis for comparing the effectiveness of different predictive models in terms of PR review outcomes. Notably, the effectiveness of our RL model will also be evaluated without any resampling, showcasing its prowess even when compared to baseline techniques that utilize sampling.

• **Naive Bayes.** In the context of this study, Naive Bayes\(^5\) serves as a baseline model to assess the effectiveness of the RL approach. Naive Bayes is commonly used as a starting point due to its simplicity and quick implementation. It is based on the Bayes theorem and assumes that all features in the dataset are conditionally independent given the class label. Mathematically, given a binary predictive model problem with two actions (acceptance and rejection of PR review outcomes), the Naive Bayes algorithm calculates the probability of a particular action (e.g., acceptance) given the observed feature values (i.e., PR characteristics). The action with the highest probability is then predicted as the final outcome\(^4\). The probability of an action given the features is calculated using the Bayes theorem:

\[
P(y|x_1, x_2, ..., x_n) = \frac{P(y) \times P(x_1|y) \times P(x_2|y) \times ... \times P(x_n|y)}{P(x_1) \times P(x_2) \times ... \times P(x_n)}
\]

Where:

- \(P(y)\) is the prior probability of class \(y\) (acceptance or rejection).
- \(P(x_i|y)\) is the conditional probability of feature \(x_i\) given class \(y\).
- \(P(x_i)\) is the probability of feature \(x_i\).

• **Random Forest.** Random Forest is a widely used ensemble learning method for binary predictive modeling\(^3\). It combines multiple decision trees to make predictions and is known for its robustness, scalability, and ability to handle complex datasets with high-dimensional features. Each decision tree in the ensemble is trained on a random subset of the training data and features, ensuring diversity and reducing overfitting. During prediction, the model combines the predictions of all trees using a voting mechanism, with the majority vote determining the final prediction. However, Random Forest is considered a black-box algorithm, as understanding its prediction rules can be challenging due to the large number of trees\(^4\).

• **XGBoost.** XGBoost or Extreme Gradient Boosting\(^\text{\cite{2017}}\), is an advanced and powerful ensemble learning algorithm used for predictive modeling. It combines multiple weak learners, usually decision trees, to form a strong predictive model. The algorithm operates iteratively, correcting errors from the previous iterations, which enhances overall accuracy and generalization.

\(^{4}\text{https://scikit-learn.org/stable/modules/naive_bayes.html}\)
One of XGBoost’s notable strengths lies in its efficient implementation, high scalability, and optimized speed. It achieves this through gradient boosting, a technique that minimizes the model’s loss function. Furthermore, XGBoost utilizes regularization methods to prevent overfitting, making it resilient to noisy data [37].

The implementations of Naive Bayes [5] and Random Forest Classifier [6] in Python utilizes the scikit-learn Python package. On the other hand, for the XGBoost implementation, we use the xgboost [7] Python library.

In order to address the challenge of imbalanced data in the dataset and to conduct a comparative analysis of various techniques with and without resampling, we utilize the following sampling methods. The choice to limit our sampling methods to these two is based on several considerations. First, both RUS and ROS are straightforward in implementation, ensuring methodological rigor. Second, they do not introduce the complexities associated with synthetic data generation, which allows for a more direct comparison between the natural dataset and the resampled datasets [49, 50, 53]. This focused approach enhances the validity of our comparative analysis and provides a clearer interpretation of the effectiveness of the RL-based model against other predictive techniques.

- **Random Under-Sampling (RUS)** This technique involves randomly removing instances from the majority class to achieve a balanced class distribution in the dataset. By reducing the number of samples in the majority class, this approach helps mitigate any bias that may arise from an imbalanced dataset, ensuring that the model is not overly influenced by the majority class [49, 53, 54].

- **Random Over-Sampling (ROS).** In this method, new instances are created by randomly duplicating samples from the minority class until it matches the size of the majority class. The objective of this approach is to balance the class distribution and alleviate the impact of class imbalance on the model’s performance. By increasing the number of samples in the minority class, the model can better learn from the underrepresented class and make more accurate predictions [49, 53, 54].

- **No Sampling (Original Dataset).** This is the baseline approach where we use the original imbalanced dataset without any sampling technique applied to it.

In this context, the **Imbalance Ratio (IR)** for binary class data represents the proportion of the number of samples in the minority class to that of the majority class [53]. Essentially, IR serves as a numerical measure to gauge the degree of imbalance present within the dataset. Quantifying the disparity between classes helps understand the data’s distribution and facilitates context-aware sampling. Remarkably, considering the IR of approximately 0.12 in the training data, both RUS (Random Under-Sampling) and ROS (Random Over-Sampling) techniques are implemented to adjust the resampling, resulting in an IR value of 0.5.

---

Chapter 4

Predicting GitHub PR Outcomes: RL Formalizations Utilizing PR Characteristics

In this chapter, we explore the application of RL formalizations using a comprehensive and extensive dataset that encompasses a wide range of PR characteristics. The chapter is structured into the following subsections to provide a systematic investigation of our approach. The first subsection, titled “Dataset” 4.1 delves into the data sources utilized and presents a detailed description of the dataset itself. Additionally, we discuss the various data preprocessing techniques employed to ensure the quality and relevance of the dataset for our RL-based analysis. By carefully curating this large-scale dataset, we can capture a variety of PR characteristics, enabling us to gain a more holistic view for predicting PR outcomes.

Moving forward, the “Methodology” Section 4.2 outlines the overall approach adopted in our RL formalizations. We provide a comprehensive description of the observation space, which defines the information available to the RL agent about the PR environment. Moreover, we clarify the action space, which covers the set of possible actions the RL agent can take during the decision-making process. Furthermore, we define the reward function, which serves as a guide for the RL agent’s learning process by incentivizing desirable behavior. Within this section, we also delve into the training phase of the model, outlining the specific techniques and algorithms employed to train the RL agent effectively. Additionally, we specify the evaluation metrics utilized to assess the performance and effectiveness of our RL-based approach.

Lastly, the “Comparative Analysis” Section 4.3 presents a comprehensive examination of experiments conducted and an in-depth comparison with baseline model as well as the existing predictive models. This comparative analysis serves as a critical component of our research, providing valuable insights into the performance and efficacy of our proposed RL-based approach.

Overall, this chapter lays the foundation for our exploration of RL formalizations and their application to PR outcome prediction, showcasing the significance of utilizing a large, comprehensive dataset.

4.1 Dataset

4.1.1 Data sources

The dataset utilized in this chapter is sourced from the work of Xunhui Zhang et al. (2021) [3]. In one of their previous works, Xunhui Zhang et al. (2020) [55] published an initial dataset derived from the publicly available GHTorrent\footnote{https://github.com/ghtorrent/ghtorrent.org/}
data dump dated 1st June 2019. This dataset comprised of 96 factors extracted from 76 research articles published between 2009 and 2019, encompassing a total of 11,230 software projects. The selection criteria for the projects included in the dataset were as follows: (i) the projects were actively developed, indicated by the presence of at least one new PR in the last three months; (ii) the projects represented a diverse set of programming languages, consisting of six different languages; and (iii) the projects varied in size and activity level. Project activity levels were measured based on the number of PRs, ranging from a minimum activity of 33 PRs to a maximum activity of 38,953 PRs. The dataset revealed that the first quartile (25% of the projects) had 55 PRs or fewer, the median or second quartile (50% of the projects) had 106 PRs or fewer, and the third quartile (75% of the projects) had 261 PRs or fewer.

During the literature review, it was observed that several studies focused on a limited number of specific projects to investigate PRs. These papers focused on investigating the PRs within a limited scope of projects, potentially limiting the generalizability of their findings. In contrast, the dataset compiled by Xunhui Zhang et al. (2021) [3] provides a substantial and comprehensive collection that encompasses a wide range of projects and diverse PR characteristics. As a result, this dataset offers greater generalizability and facilitates exploration across various contexts. By incorporating a wide range of projects and covering a multitude of PR characteristics, the dataset offers a broader perspective and facilitates the examination of PR dynamics, patterns, and trends in different software development scenarios. Ultimately, this enhances the applicability of research outcomes, as it allows for more reliable conclusions and insights into the PR process on GitHub.

Moreover, when comparing it to Gousios et al.’s dataset (2014) [1] and subsequent releases of GHTorrent, which regularly extracted and published GitHub data dumps, Zhang et al.’s dataset [3], based on the June 2019 GHTorrent data dump, exhibits notable enhancements. Specifically, Zhang et al.’s dataset [3] comprises 12 times more projects and 10 times more PRs, providing a significantly larger and richer resource for analysis and investigation.

Moreover, a significant advantage of this dataset is the meticulous effort made by the authors to address missing values in the foundational work of Xunhui Zhang et al. (2020) [55]. The authors leveraged multiple sources, including GHTorrent, the GitHub API, and source code repositories, to fill in the gaps where possible. Additionally, they introduced a new factor called same_user, which describes whether the contributor and integrator are the same users or not. This factor was not present in previous studies. While information about the same user may not appear directly valuable, it enhances the interpretation of other factors that hold significance only when the contributor and integrator are distinct individuals.

To provide a comprehensive overview of the dataset used in this study, Table 4.1 presents a detailed list of the incorporated factors. The following section will delve into the preprocessing steps undertaken to prepare and refine the dataset, ensuring its quality and suitability for the research at hand.
Table 4.1: A Comprehensive List of All the Factors in the Processed Dataset

<table>
<thead>
<tr>
<th>Developer Characteristics</th>
<th>Factor Description</th>
<th>Factor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>account_creation_days</td>
<td>no. of days from contributor’s account creation to PR creation</td>
<td>contrib/inte</td>
<td>contributor/integrator personality traits</td>
</tr>
<tr>
<td>contrib_contributor</td>
<td>% of previous contributor’s commits</td>
<td>core_member</td>
<td>core member or not</td>
</tr>
<tr>
<td>first_pr</td>
<td>whether first pull request or not</td>
<td>followers</td>
<td>no. of followers at PR creation time</td>
</tr>
<tr>
<td>perc_contributor</td>
<td>% contributor/integrator (neg:negative/pos:positive) emotion in comments</td>
<td>prev_pullreqs</td>
<td>no. of previous PRs</td>
</tr>
<tr>
<td>prior_interaction</td>
<td>no. of interactions to a project in the last three months</td>
<td>prior_review_cnt</td>
<td>no. of previous reviews in a project</td>
</tr>
<tr>
<td>same_user</td>
<td>whether same contributor and integrator or not</td>
<td>social_strength</td>
<td>fraction of team members interacting in last three months</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Project Characteristics</th>
<th>Factor Description</th>
<th>Factor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>asserts_per_kloc</td>
<td>no. of assertions per 1K lines of code</td>
<td>fork_num</td>
<td>no. of forks</td>
</tr>
<tr>
<td>integrator_activity</td>
<td>latest activity of the two most active integrators</td>
<td>language</td>
<td>programming language</td>
</tr>
<tr>
<td>open_issues</td>
<td>no. of open issues</td>
<td>open_prs</td>
<td>no. of open PRs</td>
</tr>
<tr>
<td>perc_external_contributor</td>
<td>% external PR contributions</td>
<td>pr_accept_rate</td>
<td>PR acceptance rate of project</td>
</tr>
<tr>
<td>project_age</td>
<td>no. of months from project to PR creation</td>
<td>requester_loc</td>
<td>stars</td>
</tr>
<tr>
<td>project_size</td>
<td>no. of active core team members in last three months</td>
<td>test_cases_per_loc</td>
<td>no. of test cases per 1K lines of code</td>
</tr>
<tr>
<td>test_lines_per_kloc</td>
<td>no. of test lines per 1K lines of code</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pull Request Characteristics</th>
<th>Factor Description</th>
<th>Factor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>at_tag</td>
<td>whether @ tag exists or not</td>
<td>churn_addition</td>
<td>no. of added lines of code</td>
</tr>
<tr>
<td>churn_diffusion</td>
<td>no. of deleted lines of code</td>
<td>ci_builds</td>
<td>no. of CI builds</td>
</tr>
<tr>
<td>codex</td>
<td>whether uses continuous integration or not</td>
<td>ci build perc</td>
<td>% CI build failed</td>
</tr>
<tr>
<td>comment_conflict</td>
<td>whether keyword ‘conflict’ exists in comments or not</td>
<td>contrib_comment</td>
<td>whether or not has a contributor comment</td>
</tr>
<tr>
<td>core_comment</td>
<td>whether or not has a core member comment</td>
<td>description_length</td>
<td>length of PR description</td>
</tr>
<tr>
<td>files_changed</td>
<td>no. of files touched</td>
<td>friday_effect</td>
<td>whether PR submitted on Friday or not</td>
</tr>
<tr>
<td>has_comments</td>
<td>whether PR has comments or not</td>
<td>has_exchange</td>
<td>whether or not has contributor/integrator comment</td>
</tr>
<tr>
<td>into_comment</td>
<td>whether or not has an integrator comment</td>
<td>lifetime_minutes</td>
<td>no. of minutes from PR creation to latest close time</td>
</tr>
<tr>
<td>num_code_comments</td>
<td>no. of code comments</td>
<td>num_code_comments_pos</td>
<td>no. of contributor comments</td>
</tr>
<tr>
<td>num_comments</td>
<td>no. of comments</td>
<td>num_commits</td>
<td>no. of comments</td>
</tr>
<tr>
<td>num_participants</td>
<td>no. of participants in PR comments</td>
<td>pers_pos_emotion</td>
<td>% negative emotion in comments</td>
</tr>
<tr>
<td>perc_pos_emotion</td>
<td>% positive emotion in comments</td>
<td>reopen_pr_cnt</td>
<td>no. of PRs reopened</td>
</tr>
<tr>
<td>src_churn</td>
<td>no. of lines changed (added+deleted)</td>
<td>test_churn</td>
<td>no. of lines of test code changed</td>
</tr>
<tr>
<td>test_inclusion</td>
<td>whether test case exists or not</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4.1.2 Data Pre-processing

Although the dataset provided by Xunhui Zhang et al. (2021) [3] exhibits high level of quality and comprehensiveness, our preliminary examination, involving manual inspection and data distribution graphs, uncovered unexpected values for certain factors. Notably, the most significant issue encountered was the presence of missing data values, which needed to be addressed to ensure the reliability of subsequent inferences. To address this concern, we conducted a series of data preprocessing steps using the Python programming language and leveraging the powerful capabilities of the pandas [2] and numpy [3] libraries. Our primary objective during this preprocessing stage was to retain as many relevant PR factors as possible while simultaneously preserving the large-scale nature of the dataset. By employing these preprocessing techniques, we aimed to enhance the dataset’s integrity, ultimately enabling more accurate and insightful analysis in our research.

- Within the dataset, the following factors: same-country, same-affiliation, contrib-follow-integrator, ci-test-passed, ci-first-build-status, and ci-last-build-status contained negative values. The occurrence of these negative values can be attributed to various factors such as discrepancies in metric definition or the presence of missing values. Given that the original authors had

made attempts to address missing values within the dataset, our decision instead was to replace those missing values with [Not a Number (NaN)] values in accordance with Python’s numpy library [4]. The use of NaN values allows us to explicitly identify and handle missing or undefined data points within the dataset. By representing missing values as NaN, we can distinguish them from other numerical values and apply appropriate strategies for subsequent data processing and analysis.

- During our analysis, it was observed that certain PR factors exhibited a significant number of empty or NaN values, exceeding 30% of the total values. Specifically, the factor bug-fix had 99.3% empty or NaN values, while same-affiliation, contrib-affiliation, and inte-affiliation had 93.5%, 74.4%, and 71.5% empty or NaN values, respectively. Additionally, same-country had 71.2% empty or NaN values, and ci-test-passed, ci-first-build-status, and ci-last-build-status had 54.5% empty or NaN values. Furthermore, first-response-time had 50.8% empty or NaN values, while contrib-follow-integrator and contrib-country had 43.6% and 39.7% empty or NaN values, respectively. To ensure the accuracy and reliability of our analysis, we made the decision to drop these factors from the dataset. Eliminating these factors with a high proportion of empty or NaN values allows us to focus on the remaining factors that contain more complete and reliable data.

- The factors open-diff, cons-diff, extra-diff, agree-diff, and neur-diff represent the absolute difference in personality traits between the contributor and integrator. These personality traits include openness, conscientiousness, extraversion, agreeableness, and neuroticism. However, considering that we already possess individual measurements of these personality traits for both contributors and integrators (contrib/inte-open, cons, extra, agree, neur), we opted to remove the redundant factors from the dataset. By doing so, we aim to eliminate redundancy and streamline the dataset for more efficient and meaningful analysis. Furthermore, the factors contrib-first-emo, inte-first-emo, and contrib-gender were also removed from the dataset due to the presence of approximately 20% empty or NaN values in each factor. This decision was made to maintain the integrity and preserve the large number of records within the dataset before dropping the empty or NaN values. Moreover, since there are other closely related or similar PR factors available, removing these factors with a significant number of missing values ensures that our analysis focuses on the most complete and informative aspects of the data.

- To avoid redundancy and to streamline the dataset, we made the decision to remove certain factors and consolidate others. The factors files-added and files-deleted were consolidated into a single factor called files-changed. Additionally, factors such as commits-on-files-touched, pushed-delta, has-participants, other-comment, and num-comments-con were removed from the dataset. This choice was driven by the availability of alternative factors that already capture similar or related information. For example, factors like num-commits, src-churn, and churn-addition provide insights into

---

commits on files, pushed changes, and code churn, respectively. Similarly, factors like has-exchange and has-comments cover the information previously captured by has-participants. Lastly, factors like inte-comments and contrib-comment serve as more appropriate substitutes for other-comment and num-comments-con, respectively.

- Finally, we made the decision to remove certain factors, namely ci-latency, has-tag, first-response-time, and part-num-code. The rationale behind these removals stems from either the absence of a clear metric or factor definition in the original dataset or a lack of variation among the set of unique values within these factors. With a limited range of distinct values, the usefulness of these factors for our analysis became limited. Consequently, we deemed it appropriate to remove them from the dataset to focus on more informative and varied factors.

- While feature selection techniques are often employed to identify key variables for predictive frameworks, our approach intentionally deviated from this practice. Our focus was not to compare the effects of different feature set sizes on the predictive performance of RL-based models. Rather, we sought to emphasize the adaptability of RL algorithms when working with extensive feature sets. Therefore, a large number of PR-related features have been retained in the dataset.

Following the removal of the aforementioned factors from the dataset, we proceeded to eliminate the entries containing empty or NaN values. By dropping these empty or missing entries, we aimed to ensure the integrity and reliability of the dataset.

In the subsequent stage of our data preprocessing, we focused on setting the appropriate data types for the factors within the dataset. This involved assigning the correct data types, such as floating-point values, integer values, or boolean values, to ensure consistency and compatibility across the dataset. Furthermore, we addressed the categorical data values present in the language factor. We used a label encoder to convert these categorical values into corresponding integer data values. This transformation allowed us to represent the categorical data in a numerical format, enabling easier integration with machine learning algorithms and other statistical techniques.

To establish associations between PRs and their corresponding projects within the dataset, the original dataset provided the owner-name, repo-name, and project-id as identifying information for each PR. However, in the dataset collected by Zhang et al. [3], these details were missing. In order to group the PRs based on their associated projects, we resorted to using the available project-id, creator-id, and last-closer-id fields.

After grouping the PRs using these available fields, we proceeded to remove the groups that consisted of only one associated PR. This step was taken to ensure that we focused on meaningful clusters of PRs with multiple associations to projects. As a result, we obtained a total of 118,922 distinct groups, covering a combined total of 1,306,078 PRs. This approach allowed us to capture the relationships between PRs and projects within the dataset, facilitating subsequent analysis of PR outcomes within the context of associated projects.
In the final step of our data preparation, we split the 118,922 groups into a training dataset and a testing dataset. The allocation was performed randomly, with 80% of the groups assigned to the training dataset and the remaining 20% allocated to the testing dataset. As a result, the training dataset contained a total of 1,045,883 PRs, while the testing dataset consisted of 260,195 PRs more details as provided in Table 4.2.

By splitting the dataset in this manner, we ensured the availability of separate datasets for training and testing our predictive models. The training dataset would be utilized to train the models and learn patterns and relationships within the data, while the testing dataset would serve as an independent benchmark for evaluating the performance and generalizability of the trained models.

<table>
<thead>
<tr>
<th>Data</th>
<th>Shape</th>
<th>Minority Class-0</th>
<th>Majority Class-1</th>
<th>Imbalance Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>train</td>
<td>(1045883, 72)</td>
<td>113,345</td>
<td>932,538</td>
<td>IR = 0.12</td>
</tr>
<tr>
<td>test</td>
<td>(260195, 72 )</td>
<td>28,122</td>
<td>232,073</td>
<td>IR = 0.12</td>
</tr>
<tr>
<td>dataset</td>
<td>(1306078, 72)</td>
<td>141,467</td>
<td>1,164,611</td>
<td>IR = 0.12</td>
</tr>
</tbody>
</table>

Table 4.2: Details and Distribution of PR Dataset: An 80/20 Split Analysis

As outlined in Section 3.5, the concept of Imbalance Ratio (IR) pertains to the extent of class imbalance present within the dataset. In our specific case, the target factor to be predicted is *merged-or-not*, which is a binary variable with values 0 (indicating rejection of PR) and 1 (indicating acceptance of PR). To calculate the IR, we determine the ratio of the number of samples in the minority class (the less prevalent class) to the number of samples in the majority class (the more prevalent class) within the binary class data.

Given the highly imbalanced nature of the dataset, it is crucial to consider and address the issue of class imbalance during the design phase of our predictive model. Hence, the IR serves as a valuable metric for quantifying this imbalance and plays a significant role in guiding our approach to model development.

Furthermore, our analysis delves into the exploration of relationships among the factors within the processed dataset. We began by examining the correlations between the target factor, *merged-or-not*, and the remaining factors. This initial exploration aimed to highlight potential connections and dependencies between these variables. To measure the strength of association between dichotomous factors (binary factors), we employed the Phi Coefficient, also known as the Matthews Correlation Coefficient\(^5\)\[^{56}\]. This coefficient provided a reliable measure of correlation specifically designed for binary factors. For the correlation between continuous and dichotomous factors, we calculated the Point-biserial Correlation Coefficient\(^6\)\[^{56}\], which quantifies the relationship between these distinct types of variables.

Upon analyzing the correlation values obtained in Fig. B.1 and Fig. B.2 (in Appendix B, Additional Figures), we discovered that they did not exhibit significant magnitudes, warranting a comprehensive correlation analysis. The absence of notable correlation values suggests that the factors within the dataset operate independently or possess minimal interdependencies. This finding suggests that

\(^6\)https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.stats.pointbiserialr.html
the factors considered in our analysis do not exhibit strong linear relationships or direct influences on the target factor, merged-or-not.

4.2 Methodology

The task of predicting PR outcomes can be effectively addressed by framing it as a sequential decision-making problem. As outlined in Section 3.2, in this study, the RL problem is formulated as an MDP, represented by a 5-tuple consisting of the state space ($S$), action space ($A$), transition probability ($P$), reward function ($R$), and discount factor ($\gamma$) as ($S, A, P, R, \gamma$). As such, the future state of the system depends solely on the current state and the action taken, without any dependency on the past states or actions. At each time step, the agent observes the current state of the environment, selects an action based on its policy, and receives a reward from the environment. The action taken by the agent influences the transition to the next state, and the process continues until a terminal state is reached.

In this section, we delve into the intricacies of the PR Review Markov Process framework and present the theoretical formulation of the RL-based decision model. It is important to note that our primary focus in this study is on the application of RL techniques rather than the development of new RL methodologies. As a result, we have employed an open-source RL framework called StableBaselines3, which offers state-of-the-art algorithms to support our research objectives, as outlined in Section 3.2. In order to select the most suitable algorithm from the StableBaselines3 framework, we carefully consider the compatibility of our action space and observation space with the available algorithms. This ensures that the chosen algorithm aligns effectively with the requirements of our PR outcome prediction task. To facilitate this decision-making process, we use the comprehensive list of the supported algorithms that exhibit compatibility with our specific action space and observation space as presented in Table 3.1 in Section 3.2.

Addressing the issue of data imbalance, we adopt a special solution by fine-tuning the reward function. This modification is essential to effectively handle the challenge posed by imbalanced data in the PR outcome prediction task. By making adjustments to the reward function, we can overcome the biases that may arise due to the disproportionate representation of different classes within the dataset.

4.2.1 PR Review Markov Decision Process

We conceptualize the PR outcome prediction process as a decision estimation game, wherein the RL agent is presented with a sample at each time step, and it must determine the appropriate action to take for that particular sample. The PR environment, subsequently provides the agent with an immediate reward and the next sample.

During the interaction between the agent and the environment, a positive reward is awarded to the agent when it accurately predicts the outcome of the PR review. Conversely, if the agent’s prediction is incorrect, a negative reward is assigned. Through this feedback mechanism, the agent attempts to learn an optimal behavior that maximizes the cumulative rewards obtained throughout the
interaction with the environment. Ultimately, the aim is to enhance the agent’s ability to accurately predict the PR review outcomes for a given set of samples.

In the context of this study, a sample represents a distinct state of the PR environment, encapsulating a set of PR characteristics or factors obtained from the dataset. Each row in the dataset corresponds to a unique PR state, with the column values representing specific attributes associated with that state. The action taken by the agent pertains to either accepting or rejecting the PR. The reward function, utilizing the value of the merged-or-not factor, verifies the correctness of the agent’s action and determines the associated reward.

To formalize the PR Review Markov Process framework as a sequential decision-making problem, we establish the following assumptions. Let the training dataset consist of samples denoted as $train\_data(TD) = \{(x_1, y_1), (x_2, y_2), \ldots, (x_i, y_i)\}$, where $x_j$ represents the $j^{th}$ sample, and $y_j$ represents the expected outcome of the corresponding PR review. The training dataset contains a total of $i$ samples, each comprising a unique PR state and its associated expected outcome.

4.2.2 Configurating RL Environment: State Space, Action Space, and Reward Functions for PR Outcome Prediction

Our objective is to train an agent that operates within the PR Review Markov Process. This agent will navigate through the PR states and make informed decisions based on the observed PR characteristics. To achieve this, we employ a sequential decision-making framework where:

- **State Space or Observation Space ($S$).** The state space in our PR Review Markov Process framework is determined by the training sample. At the start of training, the agent receives the first sample $x_1$ as its initial state $s_1$. Subsequently, the state $s_t$ of the environment at each step corresponds to the sample $x_t$.

  During each episode, the environment shuffles the order of the samples in the training dataset, ensuring a diverse sequence of PR states for the agent to encounter. The state space encompasses all possible PR states that the agent may encounter throughout the PR review process. Each PR state captures a unique combination of PR characteristics, reflecting the current state of the review.

  To effectively represent the PR characteristics within the state space, we utilize the **Dict** type gym space from the gym library.\(^7\) Given that our dataset encompasses as many as 72 PR-related features, the use of a **Dict** type is particularly advantageous. It allows for a flexible and structured representation of the large and diverse state space, accommodating various types of PR characteristics without constraint. This facilitates an efficient mapping of real-world PR attributes into our model, thereby enabling a more comprehensive and nuanced RL-based analysis. Excluding timestamp and identification features such as id, project-id, creator-id, last-closer-id, and last-close-time, a total of 67 PR characteristics are associated with each PR in the training dataset.

\(^7\)https://www.gymlibrary.dev/api/spaces/
The state is represented as a dictionary, where the keys correspond to the names of the PR factors, and the values associated with each key are defined using the Box type gym space. This allows us to encompass the entire range of possible values for each PR feature. For instance, some PR factors may contain floating-point values between 0 and 1, while others may have binary values of either 0 or 1. Additionally, some PR factors may take on integer values within a specific range. We take these variations into account when defining the observation space, ensuring that all possible values of the state are adequately covered.

The state space definition encompasses the full range of PR characteristics and their potential values, enabling the agent to effectively perceive and navigate the PR review environment. For a more comprehensive understanding, refer to the definition provided in Listing 1 and Listing 2, which covers all possible values of the state, along with an example of a state at step t.

Listing 1: Observation Space Specifications PR Environment - type gym.Dict

```python
# Defining the observation space
from gym.spaces import Dict, Box
observation_space = Dict({
    'account_creation_days': Box(low=np.array([0]), high=np.array([4200]), dtype='int'),
    'at_tag': Box(low=np.array([0]), high=np.array([1]), dtype='int'),
    ...
    ...
    'test_inclusion': Box(low=np.array([0]), high=np.array([1]), dtype='int'),
    'test_lines_per_kloc': Box(low=np.array([0]), high=np.array([1001]), dtype='float')
})
```

Listing 2: Sample Representation of the Observation Space in the PR Environment

```python
# Sample at stage t
# observation_space.sample() yields and Ordered Dictionary of tuples:
OrderedDict([
    ('account_creation_days', array([1525])),
    ('at_tag', array([0])),
    ...
    ...
    ('test_inclusion', array([0])),
    ('test_lines_per_kloc', array([717.90979702]))
])
```

• **Action Space (A).** The action space in our PR Review Markov Process framework encompasses the available actions that the agent can take in response to a given PR state. In this context, the agent’s actions involve deciding whether to accept or reject the PR under review.
Within the training dataset, the factor *merged-or-not* provides the expected prediction outcome of the associated PR. This factor is dichotomous in nature, with a value of 0 representing PR rejection and a value of 1 representing PR acceptance. Based on this, we define the action space as a *Discrete* type with two possible actions: Action Space = \{0, 1\}.

Given the comprehensive nature of the dataset, consisting of 72 features and over 1.3 million PRs, and considering the objective of predicting the PR outcome, we restrict the action space to these two values. The simplicity of the action space allows the agent to focus on optimizing its decision-making process and effectively predicting the outcomes of PR reviews. For a more comprehensive understanding, refer to the definition provided in Listing 3, which covers all possible values of the action space.

Listing 3: Action Space Specifications PR Environment - type gym.Discrete

```python
# Defining the action space
from gym.spaces import Discrete
action_space = Discrete(2)  # two possible values: 0 or 1 of type(): int
```

- **Episode.** It can be represented as \{(s_1, a_1, r_1), (s_2, a_2, r_2), \ldots, (s_z, a_z, r_z)\}, where \((s, a, r)\) represents a tuple consisting of the PR state, action taken by the agent, and the immediate reward received upon taking that action while \(z\) is the terminal step. In the context of this study, an episode is considered complete when predictions have been made for all the samples in the training dataset or when the agent fails to predict the rejection outcome of a PR, which is the minority outcome in the dataset.

- **Policy** \(\pi(a_t|s_t)\). The policy function \(\pi(a_t|s_t)\) is a mapping that associates each PR state \(s_t\) with an action \(a_t\) selected by the agent. In mathematical notation, it can be represented as \(\pi: S \rightarrow A\), where \(S\) is the state space and \(A\) is the action space. The policy function determines the agent’s behavior in the environment by prescribing the action to be taken in each state.

- **Reward Function** \(R(s_t, a_t)\). The reward function plays a crucial role in reinforcement learning, as it guides the agent’s learning process by providing feedback on its actions. In the initial implementation, we adopted a simple reward function that assigned a positive reward of +1 to the agent for correctly predicting the PR outcome and a negative reward of -1 for incorrect predictions after each step. This straightforward approach yielded a relatively good overall accuracy when tested on the testing dataset. However, we encountered a significant issue with this reward function—the model exhibited extreme bias and ended up accepting all the PRs in the training dataset. The root cause of this bias lies in the highly imbalanced nature of the dataset, where the majority of PRs have the target variable *merged-or-not* set to 1, indicating the acceptance of PRs. To address this problem, we decided to adopt a cost-sensitive tuning approach, which involved considering higher costs for incorrect predictions and mitigating the biases in the model.
To achieve this, we combined the cost-sensitive learning solution proposed by Victoria Lopez et al. in 2013 for classification problems [50], with the IR metric previously discussed in this study. The goal was to make the agent more sensitive to PR rejection outcomes, as predicting rejections correctly is the more challenging task due to data imbalance.

The revised reward function is defined as follows:

\[
R(s_t, a_t) = \begin{cases} 
+1, & \text{if } a_t = \text{expected\ action} \text{ and } a_t = \text{PR rejection} \\
+k, & \text{if } a_t = \text{expected\ action} \text{ and } a_t = \text{PR acceptance} \\
-1, & \text{if } a_t \neq \text{expected\ action} \text{ and } a_t = \text{PR rejection} \\
-k, & \text{if } a_t \neq \text{expected\ action} \text{ and } a_t = \text{PR acceptance}
\end{cases}
\]

Here, \( k \in [0, 1) \) is a trade-off parameter that allows us to adjust the model bias. The agent receives a high reward of +1 for correctly predicting PR rejection outcomes and a low reward of -1 for incorrectly predicting PR rejection outcomes. Similarly, the rewards for correctly or incorrectly predicting PR acceptance outcomes are adjusted with +\( k \) and -\( k \), respectively.

During the training phase, we systematically define and test a set of values for \( k \) to identify the best-performing model. By incorporating the cost-sensitive tuning approach and adjusting the reward function, we tried to improve the agent’s performance and mitigate biases, making it more effective in predicting PR outcomes accurately.

In summary, the reward function determines the reward the agent receives based on its action in a given state. The objective is to find the optimal prediction policy that maximizes the cumulative rewards. The reward function calculates the immediate reward at the end of each step, while the cumulative reward is the sum of rewards obtained over an entire episode.

### 4.2.3 Training Approach for RL-based Prediction Model

In this section, we first refer to Table 3.1 in Section 3.2 that outlines the various RL algorithms offered by the StableBaselines3 framework and shortlist the A2C, DQN, and PPO algorithms that align with the state space and action space configurations detailed in the preceding section. This selection ensures that the chosen RL algorithms are suitable for our PR outcome prediction task and will be used in our experiments for model training and evaluation.

To train the RL-based predictive model for PR reviews, we begin by constructing the RL environment based on the definition of the PR Review Markov Process framework, taking into account the complexity and the amount of training data available in the dataset. As previously mentioned, the training dataset used in this study comprises of 1,045,883 unique PRs, each characterized by 72 PR-specific features. The IR is calculated to be 0.12, reflecting the class imbalance present in the dataset. Given the structure of the training samples and the binary nature of the action space (representing rejection or acceptance of the PR), the architecture of the RL network is designed accordingly. The number of outputs is set to two, representing the discrete values of 0 or 1 for the PR review decision.
The training process follows Algorithm 1 as described below. At the beginning of each episode, the reset function is called, which shuffles all the PR samples in the training data and provides the first sample to the agent as its initial state. The agent, based on the action selected from the action space, then predicts the PR review outcome and receives the immediate reward and the next PR state in the step function. The agent repeats this action-selection process, estimating the PR review outcome for each subsequent PR sample, and receiving corresponding rewards and PR states in the step function. This iterative process continues until all the PR samples in the training dataset have been reviewed by the agent, or if it ends prematurely in the event of the agent making an incorrect prediction for the PR rejection outcome. Each episode represents a single complete review cycle while the agent receives a reward after every step.

At the start of a new episode, the RL environment is reset using the reset function, and the PR samples in the training data are reshuffled, initiating a fresh learning cycle. The entire process is then repeated for the next episode. The total number of timesteps for model learning is initially set to 1,400,000, which aligns with the total number of PRs in the dataset, ensuring that the agent experiences a comprehensive range of PR states during the training phase.

During the training process, the RL model seeks to learn an optimal behavior by maximizing the cumulative rewards obtained from its interactions with the environment. As each episode progresses, the agent refines its prediction policy based on the experienced rewards, aiming to achieve the highest possible accuracy in predicting PR outcomes for the samples in the training dataset.

In the context of the imbalanced nature of the dataset, a cost-sensitive reward function has been devised for the RL-based PR outcome prediction model. Initially, we set a range of values for the trade-off parameter \( k \), specifically \( \{0.05 \times IR, 0.5 \times IR, IR, 2 \times IR, 3 \times IR, 4 \times IR, 5 \times IR\} \), where IR is the Imbalance Ratio. For each training run and for all the RL algorithms under consideration, we perform multiple iterations, each with a different value of \( k \).

After the training process is completed, we compare the results of all the runs and select the best three runs based on their performance. These top-performing runs, each associated with an optimal value of \( k \), are then further analyzed and discussed in the results section 4.3.1. This approach enables us to identify the most suitable trade-off parameter that effectively balances the reward function and addresses the imbalanced nature of the dataset, yielding promising results for predicting PR outcomes.

### 4.2.4 Training Parameters

In the previous Section [?], which details the training process for the predictive model, we elaborated on the employment of three RL algorithms provided by StableBaselines3: DQN, A2C, and PPO. Importantly, for both the RL algorithms and the underlying neural network architectures, we use the default settings provided by StableBaselines3 to initiate the RL models. This approach ensures methodological consistency and allows for straightforward replication of our study. During the training phase, the model undergoes training using the learn() method, which facilitates the learning process based on the interactions with the PR environment and the associated rewards.
Algorithm 1: Training the RL-based PR Outcome Prediction Model

Input: Training dataset \( TD = \{(pr_1, o_1), (pr_2, o_2), \ldots, (pr_i, o_i)\} \)

/* PR State-outcome pairs, outcome is the expected action set from the merged-or-not PR factor */

Input: No. of PR samples in training data = \( i \)

Input: Total timesteps \( T = 1,400,000 \)

Output: Trained RL-based PR prediction model

1. Initialize the RL environment
2. Set the action space \( A = \{0, 1\} \) // Actions: 0 (reject), 1 (accept)
3. Set the observation space \( O \) with the selected PR characteristics as described in \( TD \)
4. Set \( step_t = 1 \) // Number of timesteps
5. Set \( E = 0 \) // Number of Episodes

while \( step_t < T \) do
    Call reset function to reset the environment for a new episode
    Set \( t = 1 \) // Step within the episode
    Set initial state \( s_1 = pr_1 \)

    repeat
        Select action \( a_t \) using RL policy \( \pi: a_t = \pi(s_t) \);
        Perform action \( a_t \) in the environment to get the next state \( s_{t+1} \) and the immediate reward \( r_t \);
        Update the RL model using the observed reward \( r_t \):
        UpdateModel(\( s_t, a_t, r_t, s_{t+1} \));
        Set \( s_t = s_{t+1} \);
        Increment \( t: t = t + 1 \);
        Increment \( step_t: step_t = step_t + 1 \);
    until \( t > i \) or \( r_t == -1 \);
    Increment \( E: E = E + 1 \);

end while

Return the trained RL-based PR prediction model
On the other hand, when evaluating the performance of the trained model, the policy is set to evaluation mode using the predict() method. In evaluation mode, no policy updates are performed, ensuring that the predictions are made based solely on the knowledge acquired during the training phase. By switching to evaluation mode, we can assess the model’s ability to generalize and make accurate predictions on unseen data without further updates to its decision-making process. The parameters for DQN object, PPO object, and A2C object are in the Appendix A, Table A.1, A.2, and A.3 respectively. The parameters for the learn() method and predict() method are in Table 4.3 and Table 4.4 respectively.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>total_timesteps</td>
<td>the total number of samples (env steps) to train on</td>
<td>1400000</td>
</tr>
<tr>
<td>callback</td>
<td>callback(s) called at every step with state of the algorithm</td>
<td>None</td>
</tr>
<tr>
<td>log_interval</td>
<td>the number of episodes before logging</td>
<td>log_interval *</td>
</tr>
<tr>
<td>th_log_name</td>
<td>the name of the run for TensorBoard logging</td>
<td>name **</td>
</tr>
<tr>
<td>reset_num_timesteps</td>
<td>whether or not to reset the current timestep number (in logging)</td>
<td>TRUE</td>
</tr>
<tr>
<td>progress_bar</td>
<td>display a progress bar using tqdm and rich</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

* log_interval =

\[
\begin{cases}
4 & \text{if using DQN} \\
1 & \text{if using PPO} \\
100 & \text{if using A2C}
\end{cases}
\]

** name = {‘DQN’, ‘PPO’, ‘A2C’}

Return type =

\[
\begin{cases}
\text{None} & \text{if DQN} \\
\text{TypeVar(SelfPPO, bound=PPO)} & \text{if PPO} \\
\text{TypeVar(SelfA2C, bound=A2C)} & \text{if A2C}
\end{cases}
\]

Table 4.3: Training Parameter and Values of the learn() Method

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>observation</td>
<td>the input observation</td>
<td>obs *</td>
</tr>
<tr>
<td>state</td>
<td>the last states</td>
<td>None</td>
</tr>
<tr>
<td>episode_start</td>
<td>the last masks</td>
<td>None</td>
</tr>
<tr>
<td>deterministic</td>
<td>whether or not to return deterministic actions</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

* input observation = obs (type: OrderedDict)

Return type: Tuple[action, obs]

Table 4.4: Parameters and Values of predict() Method

Below, the discussion includes additional algorithm-specific parameters and network architectures. Specifically, for the DQN approach, the ‘MultiInputPolicy’ class incorporates the Rectified Linear Unit (ReLU) activation function. The feature extractor is set to CombinedExtractor to flatten the observation space, which is of type Dict. The default optimizer used is Adam.

The neural network architecture, as shown in Listing 4, starts with an input layer of size 67, where the number of input features corresponds to the number of PR factors in the training dataset. The first hidden layer is a fully connected layer with 67 input features and 64 output features (or neurons). The ReLU activation function is employed in the hidden layers, determining how the weighted sum of inputs is transformed into output values from nodes in the network layer. The second hidden layer is also a fully connected layer that takes the 64 output...
features from the previous layer and generates 64 new output features. Finally, the output layer receives the 64 output features from the previous layer and predicts the outcome of PRs using a vector of 2 elements. Specifically, the model assigns a value of 0 to indicate rejection of a PR and 1 to signify acceptance of a PR.

Listing 4: Neural Network Architecture for DQN

```python
@DQN
(q_net): Sequential(  
  (0): Linear(in_features=67, out_features=64, bias=True)  
  (1): ReLU()  
  (2): Linear(in_features=64, out_features=64, bias=True)  
  (3): ReLU()  
  (4): Linear(in_features=64, out_features=2, bias=True)  
)
```

The ‘MultiInputPolicy’ class is utilized for both PPO and A2C algorithms. It employs the Hyperbolic Tangent (Tanh) activation function and the CombinedExtractor feature extractor, which is responsible for flattening the observation space of type Dict. The default optimizer used in these algorithms is Adam.

The network architecture for the ‘actor’ consists of two layers, with each layer containing 64 units. The input layer has a size of 67, corresponding to the number of input features, or PR factors, present in the training dataset. The first hidden layer is a fully connected layer, accepting 67 input features and generating 64 output features (or neurons). The activation function Tanh is applied in the hidden layers.

The second hidden layer is another fully connected layer, taking the 64 output features from the previous layer and producing 64 new output features. To align the output dimensions and activation function for discrete predictions (0 for PR rejection and 1 for PR acceptance), an additional Softmax layer is added.

Similarly, the ‘critic’ also follows a similar architecture, which means it has the same layers and units as the actor. Overall, as shown in Listing 5, both the actor and the critic in the PPO and A2C algorithms share the same network architecture, with the addition of a Softmax layer to handle the discrete
predictions.

Listing 5: Neural Network Architectures for A2C and PPO

```python
@PPO @A2C
(mlp_extractor): MlpExtractor(
  (policy_net): Sequential(
    (0): Linear(in_features=67, out_features=64, bias=True)
    (1): Tanh()
    (2): Linear(in_features=64, out_features=64, bias=True)
    (3): Tanh()
  )
  (value_net): Sequential(
    (0): Linear(in_features=67, out_features=64, bias=True)
    (1): Tanh()
    (2): Linear(in_features=64, out_features=64, bias=True)
    (3): Tanh()
  )
)(action_net): Linear(in_features=64, out_features=2, bias=True)
(value_net): Linear(in_features=64, out_features=1, bias=True)
```

**Summary.** Section 4.2.1 outlines how to formalize the PR Outcome prediction problem as an RL problem. Section 4.2.2 defines the State Space, Action Space, and the Reward Function for effective PR predictions.

### 4.3 Comparative Analysis

#### 4.3.1 Experiments

Based on the comprehensive literature review findings presented in Section 2.5, we have selected CTCPPre, RForest.PR, and their underlying methodologies, XGBoost Classifier and Random Forest Classifier, for comparison against our proposed RL-based model for PR outcome prediction. Naive Bayes will serve as the baseline for all the comparisons. In accordance with the evaluation metrics outlined in Section 3.4, the predictive performances of all the techniques will be assessed using accuracy, F1-score, G-mean score, and AUC-ROC.

### Primary Results

In the initial phase of our study, our primary aim was to identify the optimal trade-off factor (k) in the reward function for each of the three selected RL algorithms: DQN, A2C, and PPO. Table 4.7, Table 4.5, and Table 4.6 list the top-performing trade-off values for each algorithm. For this preliminary examination,
we set the parameter of total_timesteps to 1,400,000 for every algorithm. We divided the dataset using an 80/20 split to generate training and testing datasets with the following dimensions respectively: training data of (1045883, 72) and testing data of (260195, 72). This resulted in an Imbalance_Ratio(IR) = 0.12.

As a prelude to our primary results, we conducted a preliminary investigation by assigning the trade-off factor a set of values \( k = \{0.1 \times IR, 0.5 \times IR, 1 \times IR, 5 \times IR\} \) such that \( k \) is within the interval \([0, 1)\) for all three RL algorithms. Based on the results derived from training each algorithm’s implementation with the training data and subsequently evaluating their performance using the test data, we made additional adjustments to the set of trade-off factors (k values) for each algorithm, including A2C, DQN, and PPO.

### A2C Algorithm

For the A2C algorithm, our initial observations revealed certain patterns in its decision-making. When the trade-off factor was set to anything but \( k = 0.5 \times IR \), the algorithm predominantly favored the majority class action, which in our scenario corresponds to the PR acceptance (indicated by a value of 1 for the factor merged-or-not). Conversely, for \( k = 0.1 \times IR \), the algorithm leaned towards the minority class action, here symbolizing the PR rejection (signified by a value of 0 for the factor merged-or-not).

<table>
<thead>
<tr>
<th>Total Timesteps (T) = 1,400,000</th>
<th>F-1 Score</th>
<th>G-mean</th>
<th>AUC-ROC</th>
<th>Overall Accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Class-0</td>
<td>Class-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( k = 0.05 \times IR )</td>
<td>0.20</td>
<td>0.00</td>
<td>0.00</td>
<td>0.50</td>
</tr>
<tr>
<td>( k = 0.5 \times IR )</td>
<td>0.23</td>
<td>0.91</td>
<td>0.45</td>
<td>0.57</td>
</tr>
<tr>
<td>( k = 1 \times IR )</td>
<td>0.00</td>
<td>0.94</td>
<td>0.01</td>
<td>0.50</td>
</tr>
</tbody>
</table>

- 80/20 split: train.shape (1045883, 72) and test.shape (260195, 72)
- Imbalance Ratio = 0.12

Table 4.5: Performance of A2C Algorithm for PR Outcome Prediction with Different Values of Trade-Off Factor (k)

Given these insights, we adjusted the trade-off factor \( k \) values for further testing, opting for \( k = \{0.05 \times IR, 0.2 \times IR, 0.3 \times IR, 0.4 \times IR, 0.5 \times IR\} \). Table 4.5 compiles the top three results based on this set. As Table 4.5 illustrates, apart from \( k = 0.5 \times IR \), all models returned a g-mean score of 0.0. This zero score points to a complete inability to predict either the majority class action (in cases where \( k = \{0.05 \times IR, 0.1 \times IR, 0.2 \times IR, 0.3 \times IR, 0.4 \times IR\} \)) or the minority class action (in cases where \( k = \{1 \times IR, 5 \times IR\} \)).

We concluded from this observation that altering the trade-off factor did not significantly enhance A2C’s performance. Thus, we made the decision not to continue employing the A2C model for further performance analysis.

### PPO Algorithm

For the PPO algorithm, we refined our trade-off factor set to \( k = \{0.5 \times IR, 1 \times IR, 3 \times IR, 5 \times IR\} \) based on the preliminary results. The three best performances
from this revised set are presented in Table 4.6. Interestingly, we noted a pattern akin to what we observed with the A2C model. For trade-off values lower than $k = 0.5 \times IR$, the model demonstrated a bias towards minority class action, which in our study represents PR rejection (expressed as a value of 0 for the factor *merged-or-not*). Conversely, for values exceeding $k = 0.5 \times IR$, the model leaned towards the majority class action, symbolizing PR acceptance (designated as a value of 1 for the factor *merged-or-not*).

In a few instances, we achieved a G-mean value near 0.5, indicative of a generally subpar performance. This again underscores the challenges in balancing class action predictions using the PPO algorithm.

<table>
<thead>
<tr>
<th>Total Timesteps (T) = 1,400,000</th>
<th>F-1 Score</th>
<th>G-mean</th>
<th>AUC-ROC</th>
<th>Overall Accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Class-0</td>
<td>Class-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$k = 0.5 \times IR$</td>
<td>0.20</td>
<td>0.13</td>
<td>0.26</td>
<td>0.52</td>
</tr>
<tr>
<td>$k = 1 \times IR$</td>
<td>0.22</td>
<td>0.48</td>
<td>0.51</td>
<td>0.57</td>
</tr>
<tr>
<td>$k = 3 \times IR$</td>
<td>0.24</td>
<td>0.90</td>
<td>0.48</td>
<td>0.58</td>
</tr>
</tbody>
</table>

- 80/20 split: train.shape (1045883, 72) and test.shape (260195, 72)
- Imbalance Ratio = 0.12

Table 4.6: Performance of PPO Algorithm for PR Outcome Prediction with Different Values of Trade-Off Factor (k)

**DQN Algorithm**

In the final phase, we adjusted the trade-off factor for the DQN algorithm based on our preliminary findings, settling on a revised set of $k = \{1 \times IR, 2 \times IR, 3 \times IR, 4 \times IR, 5 \times IR\}$. The three best instances of performance are illustrated in Table 4.7. Interestingly, the G-mean scores for both the $k = 2 \times IR$ and $k = 3 \times IR$ cases neared 0.7, prompting us to train the model three times with these specific values. Upon review, we found $k = 2 \times IR$ to have marginally superior results with an average g-mean score of 0.68. Consequently, we decided to proceed with $k = 2 \times IR$ for further investigation with the DQN algorithm.

<table>
<thead>
<tr>
<th>Total Timesteps (T) = 1,400,000</th>
<th>F-1 Score</th>
<th>G-mean</th>
<th>AUC-ROC</th>
<th>Overall Accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Class-0</td>
<td>Class-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$k = 2 \times IR$</td>
<td>0.35</td>
<td>0.81</td>
<td>0.72</td>
<td>0.72</td>
</tr>
<tr>
<td>$k = 3 \times IR$</td>
<td>0.35</td>
<td>0.80</td>
<td>0.72</td>
<td>0.72</td>
</tr>
<tr>
<td>$k = 4 \times IR$</td>
<td>0.29</td>
<td>0.87</td>
<td>0.59</td>
<td>0.62</td>
</tr>
</tbody>
</table>

- 80/20 split: train.shape (1045883, 72) and test.shape (260195, 72)
- Imbalance Ratio = 0.12

Table 4.7: Performance of DQN Algorithm for PR Outcome Prediction with Different Values of Trade-Off Factor (k)
As the performance evaluation demonstrates, DQN delivered considerably stronger performance than the other tested algorithms for several trade-off factor values. This not only evidenced DQN’s superior predictive capability but also its stability and consistency, making it the standout choice in our comparative study.

**Summary.** DQN, A2C, and PPO were evaluated with different values of the trade-off factor $k$. The best performances are, DQN with $k = 2 \times IR$ and G-mean Score of 0.72; A2C with $k = 0.5 \times IR$ and G-mean Score of 0.45; and PPO with $k = 3 \times IR$ and G-mean Score of 0.58. DQN excels as the leading performer and will be utilized for further examination.

**Analyzing the Influence of the total_timesteps for DQN**

Building upon our primary findings, we observed that the DQN algorithm significantly outperformed its counterparts, A2C and PPO. Furthermore, we determined the optimal trade-off factor for the DQN algorithm to be $k = 2 \times IR$. Consequently, we sought to pinpoint the optimal value for the total_timesteps factor within the DQN algorithm, utilizing the aforementioned trade-off factor, for our proposed RL-based predictive model aimed at predicting PR outcomes with the comprehensive PR dataset detailed in Section 4.1.

In our initial timesteps optimization analysis, we applied an 80/20 partition to the original PR dataset, yielding a training dataset of size (1045883, 72) and a testing dataset of size (260195, 72), with an IR of 0.12. We initiated our analysis with total_timesteps set at 1,400,000, incrementally increasing this parameter by 100,000 and evaluating its performance up to a maximum of 3,500,000. Table 4.8 illustrates the top three performing timestep values. Through this analysis, we deduced that the optimal total_timesteps value for the DQN algorithm, with an 80/20 train/test data split and a trade-off factor of $2 \times IR$, was 2,300,000. While the G-mean score and AUC-ROC scores were higher at a timestep value of 2,500,000, the 2,300,000 timestep value yielded superior results in terms of the F-1 score and individual accuracy for both minority and majority class actions. As a result, we designated the total_timesteps value of 2,300,000 for further analysis.

<table>
<thead>
<tr>
<th>Total Timesteps (T)</th>
<th>F-1 Score</th>
<th>G-mean</th>
<th>AUC-ROC</th>
<th>Accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Class-0</td>
<td>Class-1</td>
<td>Overall</td>
<td>Class-0</td>
</tr>
<tr>
<td>T = 2,000,000</td>
<td>0.50</td>
<td>0.91</td>
<td>0.77</td>
<td>0.78</td>
</tr>
<tr>
<td>T = 2,300,000</td>
<td>0.53</td>
<td>0.91</td>
<td>0.83</td>
<td>0.83</td>
</tr>
<tr>
<td>T = 2,500,000</td>
<td>0.49</td>
<td>0.86</td>
<td>0.86</td>
<td>0.87</td>
</tr>
</tbody>
</table>

- 80/20 split: train.shape (1045883, 72) and test.shape (260195, 72)
- Imbalance Ratio (IR) = 0.12

Table 4.8: Performance Evaluation of DQN Algorithm with Trade-Off Factor ($2^*IR$) for Various Total Timesteps in an 80/20 Data Split Scenario
For our second timestep optimization analysis, we manipulated the original PR dataset into a 50/50 split, resulting in a training dataset of dimensions (656007, 72) and a testing dataset of dimensions (650071, 72), both maintaining an IR of 0.12. We commenced this evaluation by setting total\_timesteps to 700,000, roughly corresponding to the total count of PRs in our training dataset. We then increased this value by increments of 100,000, assessing performance variations up to a limit of 2,000,000. Table 4.9 collates the three optimal performing timestep values. As per our findings, the best total\_timesteps factor appeared to be 1,900,000 for the DQN algorithm, under a 50/50 train/test data split and a trade-off factor configured at 2 * IR.

<table>
<thead>
<tr>
<th>Total Timesteps (T)</th>
<th>F-1 Score</th>
<th>G-mean</th>
<th>AUC-ROC</th>
<th>Accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Class-0</td>
<td>Class-1</td>
<td></td>
<td>Overall</td>
</tr>
<tr>
<td>T = 1,100,000</td>
<td>0.30</td>
<td>0.76</td>
<td>0.67</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>72</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.67</td>
<td>63</td>
</tr>
<tr>
<td>T = 1,500,000</td>
<td>0.39</td>
<td>0.81</td>
<td>0.77</td>
<td>69</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.77</td>
<td>85</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>71</td>
</tr>
<tr>
<td>T = 1,900,000</td>
<td>0.42</td>
<td>0.84</td>
<td>0.79</td>
<td>74</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.79</td>
<td>86</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>73</td>
</tr>
</tbody>
</table>

- 50/50 split: train.shape (656007, 72) and test.shape (650071, 72)
- Imbalance Ratio = 0.12

Table 4.9: Performance Evaluation of DQN Algorithm with Trade-Off Factor (2*IR) for Various Total Timesteps in a 50/50 Data Split Scenario

We implemented a third timestep optimization analysis using a 20/80 split on the original PR dataset, yielding a training dataset of dimensions (260195, 72) and a test dataset of dimensions (1045883, 72) with an IR of 0.12. Initiating with total\_timesteps fixed at 300,000—approximating the total volume of PRs in the training dataset—we increased this value stepwise by 100,000 and examined performance differences up to 1,500,000. Our top three results are exhibited in Table 4.10. In this instance, we determined that the optimal total\_timesteps value was 1,400,000 for the DQN algorithm, accompanied by a 20/80 train/test data split and a trade-off factor set at 2 * IR.

<table>
<thead>
<tr>
<th>Total Timesteps (T)</th>
<th>F-1 Score</th>
<th>G-mean</th>
<th>AUC-ROC</th>
<th>Accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Class-0</td>
<td>Class-1</td>
<td></td>
<td>Overall</td>
</tr>
<tr>
<td>T = 1,000,000</td>
<td>0.26</td>
<td>0.69</td>
<td>0.62</td>
<td>57</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>69</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.62</td>
<td>55</td>
</tr>
<tr>
<td>T = 1,400,000</td>
<td>0.34</td>
<td>0.74</td>
<td>0.72</td>
<td>63</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.74</td>
<td>87</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>60</td>
</tr>
<tr>
<td>T = 1,500,000</td>
<td>0.29</td>
<td>0.83</td>
<td>0.62</td>
<td>73</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.63</td>
<td>51</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>76</td>
</tr>
</tbody>
</table>

- 20/80 split: train.shape (260195, 72) and test.shape (1045883, 72)
- Imbalance Ratio = 0.12

Table 4.10: Performance Evaluation of DQN Algorithm with Trade-Off Factor (2*IR) for Various Total Timesteps in a 20/80 Data Split Scenario

Upon scrutinizing the performance outcomes, it can be deduced that the optimal total\_timesteps value is specific to the problem at hand and significantly reliant on the size of the training dataset.
Summary. The optimum value of the total timesteps ($T$) parameter for RL-DQN with $k = 2 \times IR$ varied based on the data split; for an 80/20 split, $T = 2,300,000$; for a 50/50 split, $T = 1,900,000$; and for a 20/80 split, $T = 1,400,000$.

Impacts of Data Resampling and Comparative Analysis

In the start of this section, we established Naive Bayes as our baseline model, while Random Forest and XGBoost were set up as benchmarks for comparative predictive performance. Additionally, we highlighted two similar PR outcome predictive models from existing literature—CTCPPre [27] and Rforest_PR [26]—for comparative analysis.

For an initial comparison, we divided our PR dataset using an 80/20 split, yielding a training dataset of dimensions (1045883, 72) and a testing dataset of dimensions (260195, 72), both exhibiting an IR of 0.12. Additionally, we incorporated Random Over Sampling (ROS) and Random Under Sampling (RUS) as resampling strategies to assess the performance of both the Random Forest Classifier and XGBoost Classifier, comparing them with the other predictive models. These results are summarized in Table 4.11. Notably, with the application of ROS and RUS resampling techniques, the IR was enhanced from 0.12 in the original dataset to 0.5 in the resampled training dataset.

In the Random Over Sampling strategy, the minority class instances in the training dataset are randomly amplified. For our particular case, the original count of minority samples in the training dataset was 113,345, which, after implementing ROS, escalated to 466,269—effectively augmenting the minority class by 352,924 instances. To improve the IR to approximately 1, it would necessitate an additional 466,269 repetitions, surging the final tally of PRs in the training dataset to well over 1.8 million.

On the other hand, in the Random Under Sampling methodology, instances of the majority class in the training dataset are randomly diminished. For our dataset, the original count of majority samples was 932,538, which post implementing RUS, reduced to 226,690—a substantial reduction by 705,848 PR instances. To boost the IR to approximately 1, it would mandate a further elimination of 592,503 instances, which would result in the final PR count in the training dataset being a mere 226,690. Considering these significant impacts on the size and balance of our dataset, we opted to only enhance the IR from 0.12 to 0.5 in the training dataset. As depicted in the results Table 4.11, our proposed RL-based strategy outperforms other predictive methods for PR outcome predictions.
<table>
<thead>
<tr>
<th>Overall Accuracy (%)</th>
<th>84</th>
<th>75</th>
<th>91</th>
<th>92</th>
<th>92</th>
<th>49</th>
<th>89</th>
<th>49</th>
</tr>
</thead>
<tbody>
<tr>
<td>F-1 Score</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Class-0</td>
<td>0.53</td>
<td>0.25</td>
<td>0.43</td>
<td>0.48</td>
<td>0.48</td>
<td>0.26</td>
<td>0.56</td>
<td>0.25</td>
</tr>
<tr>
<td>Class-1</td>
<td>0.91</td>
<td>0.85</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
<td>0.61</td>
<td>0.94</td>
<td>0.61</td>
</tr>
<tr>
<td>G-mean</td>
<td>0.83</td>
<td>0.55</td>
<td>0.54</td>
<td>0.60</td>
<td>0.60</td>
<td>0.60</td>
<td>0.77</td>
<td>0.60</td>
</tr>
<tr>
<td>AUC-ROC</td>
<td>0.83</td>
<td>0.59</td>
<td>0.64</td>
<td>0.67</td>
<td>0.67</td>
<td>0.63</td>
<td>0.78</td>
<td>0.62</td>
</tr>
</tbody>
</table>

Table 4.11: Comparative Performance of Predictive Models with 80/20 Data Split and Resampling Techniques

80/20 split: train.shape (1045883, 72) and test.shape (260195, 72)

a For DQN, IR = 0.26, k = 2*IR, T = 2,300,000

* For ROS, IR = 0.5 and train.shape: (1398807, 72)

** For RUS, IR = 0.5 and train.shape: (340035, 72)

Summary. Primary Comparative Analysis with an 80/20 data split: RL-DQN outperformed all other predictive models (with or without RUS and ROS), attaining a G-mean score of 0.83. This was achieved using parameters T=2,300,000 and k = 2 * IR.

Analyzing the Impact of Training Sample Size on Predictive Model Performance

In the previous section where we investigated the optimal value for the total timesteps parameter, we considered varying proportions of the data for training and testing (80/20, 50/50, and 20/80 splits). Building upon those findings, we now extend the comparison to include the predictive performance of alternative techniques relative to our proposed RL-based approach, all under different sizes of the training and testing datasets. When we divide the PR dataset with an 80/20 split, yielding a training set of (1045883, 72) and a testing set of (260195, 72), and with an IR of 0.12, our RL-based model proves to be superior. The comparison Table 4.12 displays the results, where our proposed RL model achieves G-mean and AUC-ROC scores nearing 0.83, surpassing other prediction methods.

Next, we evaluate the model performance with a 50/50 split of the PR dataset, which gives us a training dataset of (656007, 72) and a testing dataset of (650071, 72), maintaining the same IR of 0.12. Again, as reflected in the comparison table 4.12, our RL-based approach outperforms the other methods with G-mean and AUC-ROC scores nearing 0.79.

Lastly, we apply a 20/80 split to the PR dataset, creating a training dataset of (260195, 72) and a testing dataset of (1045883, 72), keeping the IR at 0.12. The results are shown in the comparison table 4.12. Yet again, our RL-based method demonstrates superiority, outperforming the other predictive techniques with a G-mean score and AUC-ROC score approaching 0.73.
<table>
<thead>
<tr>
<th>Data Split</th>
<th>Overall Accuracy (%)</th>
<th>Class-0</th>
<th>Class-1</th>
<th>G-mean</th>
<th>AUC-ROC</th>
</tr>
</thead>
<tbody>
<tr>
<td>80/20 (a)</td>
<td>84</td>
<td>75</td>
<td>91</td>
<td>92</td>
<td></td>
</tr>
<tr>
<td>F-1 Score</td>
<td>0.53</td>
<td>0.25</td>
<td>0.43</td>
<td>0.48</td>
<td></td>
</tr>
<tr>
<td>G-mean</td>
<td>0.83</td>
<td>0.55</td>
<td>0.54</td>
<td>0.60</td>
<td></td>
</tr>
<tr>
<td>AUC-ROC</td>
<td>0.83</td>
<td>0.59</td>
<td>0.64</td>
<td>0.67</td>
<td></td>
</tr>
<tr>
<td>T = 2,300,000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50/50 (b)</td>
<td>74</td>
<td>76</td>
<td>91</td>
<td>92</td>
<td></td>
</tr>
<tr>
<td>F-1 Score</td>
<td>0.42</td>
<td>0.25</td>
<td>0.40</td>
<td>0.48</td>
<td></td>
</tr>
<tr>
<td>G-mean</td>
<td>0.79</td>
<td>0.55</td>
<td>0.52</td>
<td>0.59</td>
<td></td>
</tr>
<tr>
<td>AUC-ROC</td>
<td>0.80</td>
<td>0.59</td>
<td>0.63</td>
<td>0.67</td>
<td></td>
</tr>
<tr>
<td>T = 1,900,000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20/80 (c)</td>
<td>63</td>
<td>76</td>
<td>91</td>
<td>91</td>
<td></td>
</tr>
<tr>
<td>F-1 Score</td>
<td>0.34</td>
<td>0.26</td>
<td>0.37</td>
<td>0.47</td>
<td></td>
</tr>
<tr>
<td>G-mean</td>
<td>0.72</td>
<td>0.55</td>
<td>0.49</td>
<td>0.59</td>
<td></td>
</tr>
<tr>
<td>AUC-ROC</td>
<td>0.74</td>
<td>0.59</td>
<td>0.62</td>
<td>0.67</td>
<td></td>
</tr>
<tr>
<td>T = 1,400,000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\(a\) train.shape (1045883, 72) and test.shape (260195, 72) while IR = 0.12

\(b\) train.shape (656007, 72) and test.shape (650071, 72) while IR = 0.12

\(c\) train.shape (260195, 72) and test.shape (1045883, 72) while IR = 0.12

* For DQN, IR = 0.12 and \(k = 2 \times \text{IR}\), Total Timesteps = \(T\)

Table 4.12: Comparison of Impact of Training Sample Size on Predictive Model Performance

It is important to note that in certain scenarios, the F1-scores for individual classes rendered by the Random Forest and XGBoost classifiers slightly surpass those of our proposed RL-based PR outcome prediction method. This occurs even though these classifiers display considerably lower G-mean and AUC-ROC scores. This observation is not uncommon when working with imbalanced datasets \([50]\). As explained in our previous discussion in Section 3.4 “Evaluation Metrics”, the F1-score is a valuable metric for capturing the balance between precision and recall. In contrast, the G-mean score aims to maximize accuracy across both the majority and minority action classes, considering both sensitivity and specificity. Therefore, as our RL-based model demonstrates a higher G-mean score, it indicates a more balanced performance in handling both the PR acceptance and rejection actions. To further validate our findings, please refer to the performance
results displayed in Table 4.13.

In the case of a 50/50 training and testing data split, the testing dataset is sized (650071, 72). After the RL model and other predictive techniques have been trained, the testing phase further divides this dataset into two groups based on the *merged-or-not* factor. In other words, we segment the testing data into the underlying majority and minority class actions (0 for PR rejection and 1 for PR acceptance). We then employ the RL model and the other predictive techniques to anticipate the outcomes of PRs in these two distinct groups, and we have summarized the results in Table 4.13.

Similarly, for a 20/80 training and testing data split, the testing dataset is sized (1045883, 72). Following the training phase for the RL model and other prediction techniques, we again divide the testing dataset into two groups, with the first group containing all the PRs associated with the minority class action of 0 (PR rejection) and the second group containing the PRs associated with the majority class action of 1 (PR acceptance).

<table>
<thead>
<tr>
<th>Data Split</th>
<th>RL-DQN(^a)</th>
<th>Random Forest Classifier</th>
<th>XBGoost Classifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>T = 1,900,000</td>
<td>86</td>
<td>27</td>
<td>5</td>
</tr>
<tr>
<td>Minority Group (Class-0)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Majority Group (Class-1)</td>
<td>73</td>
<td>99</td>
<td>95</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Split</th>
<th>RL-DQN(^b)</th>
<th>Random Forest Classifier</th>
<th>XBGoost Classifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>T = 1,400,000</td>
<td>87</td>
<td>24</td>
<td>5</td>
</tr>
<tr>
<td>Minority Group (Class-0)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Majority Group (Class-1)</td>
<td>60</td>
<td>99</td>
<td>95</td>
</tr>
</tbody>
</table>

\(^a\) train.shape (656007, 72) and test.shape (650071, 72) while IR = 0.12
\(^b\) train.shape (260195, 72) and test.shape (1045883, 72) while IR = 0.12
\(^*\) For DQN, IR = 0.12 and k = 2*IR, Total Timesteps = T

Table 4.13: Evaluation of Predictive Models for Special Test Cases: Highlighting the Importance of G-mean Score

Table 4.13 reveals that our RL-model outperforms all other prediction techniques by a substantial margin, despite its slightly lower F1-score. This result emphasizes the significance of the G-mean score, at least within the context of our present study on PR outcome prediction utilizing an imbalanced dataset. It is noteworthy that in every instance, our proposed RL-based approach significantly excels in predicting PR rejection outcomes.
Summary.

- DQN outperforms other predictive models across various data splits (80/20, 50/50, 20/80), achieving the best G-mean score in each split with values of 0.83, 0.79, and 0.72 respectively, as detailed in Table 4.12.

- DQN significantly excels in predicting PR rejection outcomes, offering practical benefits in real-world scenarios, especially with dedicated automated pipelines for PR merges. This ability to anticipate PR rejection can be particularly valuable in such contexts.
Chapter 5

RL Formalizations Utilizing PR Discussions

In this chapter, we delve into the application of RL formalizations focusing exclusively on PR discussions on GitHub. Our aim is to provide a unique perspective for predicting PR outcomes using RL, which complements the RL formalization proposed in the previous chapter.

Comments are integral to the GitHub PR review process, enabling dynamic discussions among developers, reviewers, and stakeholders. They serve as a platform for feedback, clarifications, and suggestions, significantly impacting the decision to accept or reject a PR. Developers provide explanatory comments detailing their changes, while reviewers carefully analyze the code and offer constructive feedback, address concerns, and propose changes. By incorporating the raw PR discussions into our RL formalization for outcome prediction, the agent gains insights into opinions and sentiments, enabling more informed decisions and improved PR outcome predictions.

The chapter is organized into distinct subsections to ensure a systematic investigation of our approach. The first subsection, titled “Dataset” 5.1, provides detailed insights into the data sources we utilized and offers a comprehensive description of the dataset itself. Additionally, we discuss the data preprocessing techniques employed to ensure the dataset’s quality and relevance for our RL-based analysis. Moving forward, the “Methodology” Section 5.2 outlines our overall approach in this second RL formalization. We describe the observation space, which defines the information available to the RL agent about this new PR environment. Additionally, we clarify the action space, which encompasses the set of possible actions the RL agent can take during the decision-making process. Furthermore, we define the reward function which is a crucial guide for the RL agent’s learning process. Within this section, we also delve into the training phase of the model, highlighting the specific techniques and algorithms employed to effectively train the RL agent. Additionally, we specify the evaluation metrics utilized to assess the performance and efficacy of our RL-based approach.

The “Experiments” Section 5.3.1 presents a comprehensive examination of the experiments conducted and an in-depth comparison with baseline models. Notably, the PR Review Markov decision process and the overall formalization in this chapter share similarities with those presented in Chapter 4. However, in this second formalization, we introduce significant changes in the observation space, episodes (and episode lengths), and the way the agent perceives the PRs in the dataset.
5.1 Dataset

5.1.1 Data sources

On GitHub, PR-related comments can be categorized into three types: comments on the PR itself, comments on specific lines within the PR (referred to as in-line code comments), and comments on specific commits within the PR (commit comments). Through our survey analysis in Chapter 6, it became evident that in-line code comments garnered substantial preference among the survey participants, comprising of software developers and software engineers. As a result, we decided to create a comprehensive dataset that specifically captures the in-line code comments within the PRs. This dataset serves as the foundation for our PR outcome prediction analysis.

Obtaining, cleaning, and ensuring the integrity of raw in-line code comments on a large scale is a challenging task. Fortunately, we were able to source a reliable data repository for such comments, as presented in the work by Akshay Sinha et al. (2021) [57]. They made use of GHArchive, an open-source project that captures and archives the public GitHub timeline, making it accessible for further analysis. By utilizing the GitHub PR Review Comment Event, the authors successfully constructed a dataset of raw comments. This data spans from January 2015 to December 2020, comprising of a substantial 54,021,838 PR review comments.

The data collected from GHArchive was specifically tailored to include in-line code comments, as indicated by the utilization of the PR Review Comment Event. According to the GitHub API documentation, the PR Comment Event object pertains to activities related to PR review comments in the PR’s unified diff. Accessing this information is only facilitated through the “pulls” GitHub REST API endpoint. To ensure data quality and consistency, the final version of the cleaned dataset was published in JSONLines format, containing exclusively English language comments. Notably, the dataset included both the raw comments and their corresponding commit-id.

To enrich the dataset with additional valuable information, we utilized the commit-id in combination with the GitHub REST API. Through this process, we augmented the dataset with features such as owner-name, repo-name, pull-no, created-at, and merged-or-not. Detailed explanations of all the features in the curated PR Comments Dataset are provided in Table 5.1.

5.1.2 Feature Extraction and Sentiment Analysis of PR comments

Upon gathering essential project characteristics such as owner-name, repo-name, pull-no, created-at, and key PR characteristics like merged-or-not using the GitHub REST API, we augmented the existing dataset of only raw comments (body) with additional features. This process resulted in the creation of a specialized PR Comments Dataset, specifically tailored for our second RL formalization. To

---

1https://docs.github.com/en/rest/guides/working-with-comments
2https://www.gharchive.org/
3https://docs.github.com/en/webhooks-and-events/events/github-event-types
delve deeper into the comments and extract more meaningful insights, we decided to employ sentiment analysis. This approach allowed us to quantitatively assess the emotions associated with each comment, categorizing them as negative, neutral, or positive. By incorporating the sentiment scores, we aimed to create predictive models for further analysis, adding another layer of understanding to the raw comments in the dataset.

Additionally, we extracted several other relevant features from the raw comments to capture the complete context and nuances expressed within each comment. These extracted features, along with the sentiment scores, contribute significantly to the overall richness and comprehensiveness of the dataset.

In the following sections, we will elaborate on the details of the sentiment analysis approach adopted, explaining how we transformed the raw comments into quantifiable sentiment scores. Furthermore, we will delve into the process of extracting the various contextual features from the comments, which will provide a detailed overview of the dataset and set the stage for our subsequent analysis and predictive modeling efforts.

<table>
<thead>
<tr>
<th>Factor</th>
<th>Description</th>
<th>Factor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner_name</td>
<td>the account owner of the repository (not case sensitive)</td>
<td>repo_name</td>
<td>the name of the repository without the .git extension (not case sensitive)</td>
</tr>
<tr>
<td>pull_no</td>
<td>the number to identify the PR</td>
<td>merged_or_not</td>
<td>whether PR has been merged or not</td>
</tr>
<tr>
<td>body</td>
<td>describes the actual comment</td>
<td>created_at</td>
<td>time at which the comment was created</td>
</tr>
<tr>
<td>word_count</td>
<td>no. of words in the comments</td>
<td>stopw_ratio</td>
<td>ratio of no. stopwords to total word count in the comment</td>
</tr>
<tr>
<td>neg_vr</td>
<td>negative polarity score of the comment</td>
<td>neu_vr</td>
<td>neutral polarity score of the comment</td>
</tr>
<tr>
<td>pos_vr</td>
<td>positive polarity score of the comment</td>
<td>compound</td>
<td>overall polarity score of the comment</td>
</tr>
<tr>
<td>has_code_element</td>
<td>whether the comment makes a code suggestion or not</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.1: A Comprehensive List of Features in the Curated PR Comments Dataset

**Feature Extraction**

- **has-code-element**: During the manual inspection and initial exploration of the raw comments (body field in the dataset), we noticed that some developers engaged in discussions related to entire code segments or specific sections of code. This observation was possible because the comments were in-line code comments, allowing developers to refer to specific lines of code. In many cases, these developers provided actual code suggestions directly within the comments. As GitHub supports markdown in the comments section, identifying such code suggestions from the remaining text in the raw comments became feasible.

To systematically identify and capture these code suggestions in the raw comments, we devised a Regular Expression (regex) approach. This approach allowed us to create a new feature called *has-code-element*, which serves as a binary indicator. When a comment contains a code suggestion, the feature is assigned a value of 1; otherwise, it is set to 0. The implementation of this process was achieved using the `re` package in Python, enabling us to efficiently examine all the comments in the dataset.
• **word-count**: To quantify the comments and prepare them for the predictive models, we decided to calculate the total word count for each comment. This word count considers only valid English words, excluding punctuations, symbols, whitespace, and other special characters. To ensure the validity of the English words, we utilized the *PyEnchant* Python package[^4] which offers ready-to-use dictionaries for various English versions, such as *en-GB* (English British) and *en-US* (American English). These dictionaries are based on the Hunspell library, a widely used free spell checker and morphological analyzer licensed under LGPL/GPL/MPL tri-license[^5].

The *PyEnchant* package, in conjunction with the Hunspell library, provides a reliable means to verify the English words in the comments, ensuring their accuracy and relevance for further analysis. This step is particularly important to guarantee that the word counts accurately represent the linguistic content of the comments, enabling us to gain meaningful insights from the data. The availability and widespread use of the Hunspell library in well-known open-source tools, such as LibreOffice, Mozilla Firefox, Google Chrome, and multiple Linux distributions, further validate its suitability for our analysis of GitHub PR comments.

• **stopw-ratio**: During the sentiment analysis process, it is crucial to identify and eliminate stop words from the sentences. Stop words are common words such as “the,” “so,” “are,” “is,” etc., which are deemed insignificant for the natural language data processing. To achieve this, we utilized the *nltk* library[^6] for Python, which provides convenient tools for natural language processing tasks.

While sentiment analysis typically discards stop words, we were interested in investigating whether their presence or absence would impact the performance and effectiveness of our predictive model. Therefore, we calculated the stop word ratio for each comment in the dataset. The stop word ratio represents the proportion of stop words in the comment compared to the total number of words in the comment.

\[
\text{Stop Word Ratio} = \frac{N_{stop}}{N_{total}} \quad \text{where, } N_{stop} = \text{count of stop words in the comment and } N_{total} = \text{count of total words in the comment.}
\]

**Sentiment Analysis**

To conduct sentiment analysis on the comments in our dataset, we employed the Valence Aware Dictionary and Sentiment Reasoner (VADER) as our primary tool as outlined in Chapter 3, Section 3.3[^3].

After conducting sentiment analysis with VADER, we enriched our dataset by adding specific features. For each comment, we included individual emotion scores: the positive emotion score, labeled as ‘pos-vr’; the negative emotion score, tagged as ‘neg-vr’; and the neutral emotion score, denoted as ‘neu-vr’. In addition to these individual emotion scores, we also added a ‘compound’ score, which

[^4]: https://pyenchant.github.io/pyenchant/
[^5]: https://github.com/hunspell/hunspell
[^6]: https://www.nltk.org/
encapsulates the overall sentiment of the comment as previously explained. To assess the effectiveness of VADER, we implemented the following procedures:

- Depending on the value in the \textit{compound} attribute, we introduced a temporary categorical feature called \textit{emotion} to the dataset. This \textit{emotion} feature can take one of three possible values: 0 for negative emotion, 1 for neutral emotion, and 2 for positive emotion. The classification into these categories was determined by the following thresholds in the ‘compound’ score \cite{48}:

\[
\text{emotion} = \begin{cases} 
0, & \text{if } \text{compound} \leq -0.05 \\
1, & \text{if } -0.05 < \text{compound} < 0.05 \\
2, & \text{if } \text{compound} \geq 0.05 
\end{cases}
\]

- Upon introducing the temporary ‘emotion’ feature to the dataset, we took a random selection of 200 comments for further analysis. To ensure a balanced class distribution, we selected 70 comments marked as negative (where \textit{emotion} is set to 0), 65 comments marked as neutral (where \textit{emotion} is set to 1), and 65 comments marked as positive (where \textit{emotion} is set to 2).

Following this selection process, we manually reviewed all 200 comments. During this review, we classified each comment based on its overall sentiment, categorizing it as positive (2), neutral (1), or negative (0). This manual classification enabled us to establish a standard against which we could compare the automated sentiment classification performed by VADER.

- By contrasting the VADER-assigned classifications with our manually determined categories, which were obtained through a careful review of the comments, we were able to generate a classification report as shown in Table 5.2. This comparative analysis revealed a G-mean score of approximately 0.75, reflecting the balanced accuracy of the classification. Moreover, the overall accuracy rate of the sentiment categorization performed by VADER was found to be around 74%. While we made every effort to be objective in our manual classification, it’s worth noting that there may still be a degree of subjective bias. Nonetheless, this level of performance provided us with confidence in the validity of our sentiment analysis and its capacity to support our broader research goals.

<table>
<thead>
<tr>
<th></th>
<th>precision</th>
<th>recall</th>
<th>f1-score</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.83</td>
<td>0.75</td>
<td>0.79</td>
<td>77</td>
</tr>
<tr>
<td>1</td>
<td>0.77</td>
<td>0.67</td>
<td>0.71</td>
<td>75</td>
</tr>
<tr>
<td>2</td>
<td>0.63</td>
<td>0.85</td>
<td>0.73</td>
<td>48</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>accuracy</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.74</td>
<td></td>
<td>200</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>macro avg</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.74</td>
<td>0.76</td>
<td>0.74</td>
<td>200</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>weighted avg</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.76</td>
<td>0.74</td>
<td>0.75</td>
<td>200</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>G-mean Score</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.75416</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.2: VADER Performance Analysis: Classification Report and G-mean Score
Upon feature extraction and sentiment score computation for the raw comments, we scrutinized the dataset for any empty or missing values. Due to our meticulous data curation process, which utilized GitHub REST APIs and a clean, comprehensive data source for raw comments, we did not find any such discrepancies in the dataset.

Remarkably, our final dataset comprised of 588,097 comments associated with 66,281 different PRs and incorporated 15 distinct PR characteristics. In order to facilitate model development and evaluation, this dataset was divided into a training set and a testing set using an 80/20 split. Specific details about this division are provided in Table 5.3.

<table>
<thead>
<tr>
<th>Data</th>
<th>Shape</th>
<th>Count Minority Class-0</th>
<th>Count Majority Class-1</th>
<th>Imbalance Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>train</td>
<td>(471831, 15)</td>
<td>100,454</td>
<td>371,377</td>
<td></td>
</tr>
<tr>
<td>test</td>
<td>(116266, 15)</td>
<td>24,484</td>
<td>91,782</td>
<td>IR = 0.26</td>
</tr>
<tr>
<td>dataset</td>
<td>(588097, 15)</td>
<td>124,938</td>
<td>463,159</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.3: Details and Distribution of PR Comments Dataset: An 80/20 Split Analysis

Additionally, we conducted an exploratory study to investigate the correlation between the merge-or-not feature and other attributes associated with PR comments. To assess these relationships, we utilized two statistical measures: the Phi Coefficient, which gauges correlation between dichotomous (binary) features, and the Point-Biserial Correlation Coefficient, which measures correlation between continuous and dichotomous features. For our analysis, the continuous features under consideration were pos-vr, neg-vr, neu-vr, compound, stopw-ratio and word-count. However, it is important to note that the results we obtained from these correlation calculations did not suggest strong relationships, as outlined in Fig. B.3 (in Appendix B, Additional Figures). The magnitudes of the correlation coefficients were not large enough to signify substantial or meaningful associations between merge-or-not and the various PR comment-related features. Therefore, we did not find it necessary to further delve into a more detailed correlation analysis.

5.2 Methodology

The methodology implemented in this second RL formalization shares its foundations with the principles discussed in Chapter 4. Much like the Markov Process framework explicated in that chapter, we propose a modified PR Review Markov Decision Framework specifically for the PR Comments Dataset. We continue to perceive the task of PR outcome prediction as a Markovian sequential decision-making process, where the next state of the system relies solely on the current state and action undertaken and is independent of past states or actions. Recall that the Markov decision process is encapsulated by a 5-tuple, denoted as \((S, A, P, R, \gamma)\), which respectively represent the state space, action space, transition probability, reward function, and discount factor.

Similar to the first formalization, the open-source RL framework, Stable-Baselines3, is employed for the second one as well. Our comparative analysis
and experiments in Chapter 4 indicated that the Deep Q-Network (DQN) provided the best performance when the reward function was set such that the reward of the majority class (1 or majority action of PR acceptance) was fixed to $k = 2 \times \text{Imbalance Ratio} (IR)$. Given that this RL formalization also bears its roots in the framework discussed in Chapter 4, the observation space and action space are once again identified as Dict type and Discrete type respectively, as per the gym Python library.

Additionally, since the PR Comments Dataset is inherently imbalanced, the reward function in this RL formalization is also cost-sensitive, incorporating the IR in its specifications. We will begin with testing DQN, Proximal Policy Optimization (PPO), and Advantage Actor-Critic (A2C) algorithms, setting the reward for the majority action to $k = 2 \times IR$. Although the other algorithms are tested, we expect DQN to be the best performer (based on the results of Chapter 4).

In the following sections, we will clarify the differences in the framework, state space, action space, reward function, and episodes, starting with an overview of the updated PR Review Markov Process.

5.2.1 PR Review Markov Process for PR Comments Dataset

We devise the PR outcome prediction process as a strategic decision-making game in which the RL agent encounters a series of inline comments associated with a single PR within one episode. The timestamps of the comments dictate the sequence of review stages. At each stage, the agent receives a state from the PR environment which represents the current review stage, reflected by a specific comment. The agent’s task is to decide the appropriate action for the overall PR at that stage, i.e., to accept or reject the PR based on the given comment.

Upon each action, the PR environment offers an immediate reward of zero, signifying neutrality towards the action taken, and presents the next state, representing the subsequent comment in the review sequence. The agent then continues to determine the appropriate action based on the new state. This process repeats for every inline comment until the PR review is complete, marking the end of an episode. The conclusion of an episode sees a final PR review outcome based on the totality of actions taken by the agent. At this juncture, the PR environment provides the agent with a non-zero reward reflecting the accuracy of the agent’s overall decision and supplies the next sequence of states, or inline comments, pertaining to the upcoming PR.

Throughout the agent-environment interaction, the reward system operates on the basis of the agent’s estimation accuracy. Accurate outcome predictions earn the agent a positive reward, while incorrect predictions lead to a negative reward. This feedback mechanism encourages the agent to refine its behavior iteratively to maximize the cumulative rewards gained during its interactions with the environment. The end goal is to improve the agent’s capacity to precisely predict PR outcomes after reviewing each associated inline comment.

To summarize, a data entry from the PR Comments Dataset, in this context, symbolizes a unique collection of states from the PR environment. Each collection represents a unique PR, with individual states within the collection signifying the review stage for each inline comment associated with that PR. The action of the
agent involves deciding to either accept or reject the PR at each review stage, i.e., for each inline comment. The reward function utilizes the value of the merged-or-not factor to verify the correctness of the agent’s decision, determining the corresponding reward accordingly.

To adapt the revised PR Review Markov Process framework as a sequential decision-making problem for the PR Comments Dataset, we posit the following assumptions. We represent the training dataset as $trainData(TD) = \{(x_1, y_1), (x_2, y_2), \ldots, (x_i, y_i)\}$, where each $x_j$ signifies the $j^{th}$ PR in the dataset, and each $y_j$ indicates the expected outcome of the respective PR review. This dataset houses a total of $i$ PRs, each composed of a unique sequence of PR review stages and the corresponding expected outcome.

Therefore, $x_j$ can be depicted as a sequence of PR review stages, $x_j = \{c_1, c_2, \ldots, c_n\}$, where $c_n$ denotes the review stage for the $n^{th}$ inline comment in the $j^{th}$ PR. This encapsulates the essence of our RL formalization for the PR Comments Dataset, facilitating an efficient and insightful decision-making process.

### 5.2.2 Configurating RL Environment

Our aim is to train an artificial agent to function within the suggested PR Review Markov Process, transitioning through the various states of PR and making astute determinations based on the features of the observed PR comments. To achieve this, we employ a sequential decision-making framework where:

- **State Space or Observation Space ($S$).** Our modified PR Review Markov Process framework defines its state space based on the characteristics of PR comments in the training dataset. At the onset of training, the agent is provided with an initial state set that represents the first PR from the training dataset, which is denoted as $x_1$. This first PR, $x_1$, consists of a collection of review stages associated with each in-line code comment in the PR and is expressed as $x_1 = \{c_1, c_2, \ldots, c_n\}$. Here, $c_n$ signifies the review stage of the $n^{th}$ in-line comment in that PR.

The review stage of each comment, such as $c_1$, encapsulates various comment attributes, including word-count, stopw-ratio, compound, pos-vr, neg-vr, neu-vr, has-code-element, and merged-or-not. The agent takes this as its initial state $s_1$. As the training progresses, the agent navigates through different states. For each subsequent step, the agent is presented with a new state, denoted as $s_t$. This state $s_t$ corresponds to the review stage of the in-line code comment $c_t$. Hence, at each step, the agent updates its understanding based on the features of the comment at that particular review stage, thereby continually learning and improving its ability to predict PR outcomes.

Each PR state encapsulates a unique combination of PR comment attributes, signifying the current status of the review. Therefore, the state space should be broad enough to include all the possible PR review states that the agent might come across during the PR review process.

To effectively implement this, we utilize the Dict type gym space from the Python gym library. The state is represented as a dictionary, where the
keys are the names of the PR comment factors, and their associated values are defined using the Box type gym space. This representation enables us to cover the full spectrum of possible values for each PR comment attribute. For example, sentiment factors contain floating-point values within the range of 0 and 1 or -1 and 1, while some have binary values of 0 or 1. Moreover, some PR comment factors may contain integer values within a specific range. While defining the observation space, we accommodate these variations, ensuring comprehensive coverage of all possible state values. For a more in-depth understanding, please refer to the definition provided below in Listing 6 that covers all potential state values.

Listing 6: Observation Space Specifications - type gym.Dict

```
# Defining the observation space
from gym.spaces import Dict, Box
observation_space = Dict({
    'compound': Box(low=np.array([-1]), high=np.array([1]), dtype='float'),
    'has_code_element': Box(low=np.array([0]), high=np.array([1]), dtype='int'),
    'merged_or_not': Box(low=np.array([0]), high=np.array([2]), dtype='int'),
    'neg_vr': Box(low=np.array([0]), high=np.array([1]), dtype='float'),
    'neu_vr': Box(low=np.array([0]), high=np.array([1]), dtype='float'),
    'pos_vr': Box(low=np.array([0]), high=np.array([1]), dtype='float'),
    'stopw_ratio': Box(low=np.array([0]), high=np.array([1]), dtype='float'),
    'word_count': Box(low=np.array([0]), high=np.array([1600]), dtype='int')
})
```

- **Action Space** ($A$). The action space within our revised PR Review Markov Process framework embodies the possible actions the agent can take in response to any given state. In this setting, the agent is tasked with making a binary decision—either to accept or reject the PR currently under review. The desired action outcome for each PR in the training dataset is indicated by the `merged-or-not` attribute. This binary factor takes the value of 0 when the PR is rejected and 1 when the PR is accepted.

As such, we describe the action space as a Discrete type, meaning it includes two possible actions: 0 or 1. The definition of this action space mirrors the one discussed in Chapter 4. The definition in Listing 7 provides a thorough overview of the possible values of the action space:

Listing 7: Action Space Specifications - type gym.Discrete

```
# Defining the action space
from gym.spaces import Discrete
action_space = Discrete(2) # two possible values: 0 or 1 of type(): int
```

- **Transition Function** ($P$), **Discount Factor** ($\gamma$), and **Policy** $\pi(a_t|s_t)$. The transition function, discount factor, and policy function in the second RL formalization adhere to the same definitions established in the first RL formalization, as outlined in Section 4.2.2.
• **Episode.** Differing from the initial RL formalization presented in Chapter 4 within the scope of this study, an episode is deemed concluded once the PR outcome has been estimated for every PR review stage symbolized by the in-line code comments associated with that specific PR. Throughout an episode, the agent engages with the environment by sequentially examining the PR review stages (depicted by in-line code comments), arranged in order according to the respective timestamps of the comments, making decisions, and obtaining rewards. Consequently, during a single episode, the agent observes only one PR.

• **Reward Function** Reward Function $\mathcal{R}(s_t, a_t)$. This RL formalization introduces a slightly revised reward function compared to the one presented in Chapter 4. Although the changes have been made, the foundation of this new reward function still stems from the principles outlined in the reward function from Chapter 4. In our pursuit to enhance the agent’s sensitivity towards PR rejections, which is a more challenging task due to the data imbalance, we have updated the reward function which is defined as follows:

$$\mathcal{R}(s_t, a_t) = \{0, \forall a_t\}$$

If episode_done = True

$$\mathcal{R}(s_{termnl}, a_{effctv}) =$$

\[
\begin{cases} 
+1, & \text{if } a_{effctv} = \text{expected_action and } a_{effctv} = \text{PR rejection} \\
+k, & \text{if } a_{effctv} = \text{expected_action and } a_{effctv} = \text{PR acceptance} \\
-1, & \text{if } a_{effctv} \neq \text{expected_action and } a_{effctv} = \text{PR acceptance} \\
-k, & \text{if } a_{effctv} \neq \text{expected_action and } a_{effctv} = \text{PR rejection}
\end{cases}
\]

For any given step ‘t’ and current state $s_t$, the reward is consistently 0, regardless of the agent’s actions at the individual PR review stages (represented by the in-line code comments). However, at the episode’s conclusion, when the last observed state is the terminal state $s_{termnl}$ and the agent selects the effective action for the entire PR ($a_{effctv}$), the agent receives a non-zero reward.

More specifically, at the end of the episode (or the PR review), the agent receives a reward of +1 if it accurately identifies the final effective outcome of the PR as a rejection. In contrast, an inaccurate prediction of PR rejection results in a reward of -1. For PR acceptance outcomes, the reward is $+k$ for a correct prediction and $-k$ for incorrect predictions.

Here, the variable $k \in [0, 1)$ is again a trade-off parameter that helps fine-tune the model’s bias. Unlike in Chapter 4, we no longer systematically test various ‘k’ values during the training phase. Instead, we set $k$ to $2*IR$, based on the results obtained in Chapter 4 where the optimal model used the DQN algorithm with $k$ set to $2*IR$.

### 5.2.3 Training Approach

Drawing upon the findings from Chapter 4, our attention primarily leans towards the DQN algorithm (which demonstrated superior performance), but we also
experiment with the A2C and PPO algorithms provided by the StableBaselines3 framework in Python. The state and action space are of the Dict and Discrete types respectively, ensuring compatibility with DQN, A2C, and PPO.

In order to train the RL-based predictive model for PR reviews, we initiate by crafting an RL environment that adheres to the redefined PR Review Markov Process framework, factoring in the intricacy of the data and the volume of training data accessible in the dataset. As previously noted, the training dataset in this research includes 471,831 in-line code comments associated with 53,000 unique PRs. The IR, calculated at 0.26, mirrors the class imbalance inherent in the dataset. Considering the binary nature of the action space (depicting either rejection or acceptance of the PR) and the shape of the training samples, the structure of the RL network is designed correspondingly. The number of outputs is configured to two, symbolizing the discrete values of 0 or 1 for the PR review decision.

The training procedure adheres to Algorithm 2 as outlined below. At the start of each episode, the reset function is invoked, which randomizes the order of all PRs (each PR being a collection of PR review stages depicted by in-line code comments) within the training data and offers the first PR to the agent. For the chosen PR, the associated PR review stages and individual in-line code comments are reordered in accordance to their timestamps, with the first PR review stage (in-line code comment) in this reordered collection established as the initial state. The agent then predicts the overall PR review outcome at that stage, based on the action selected from the action space, and is given an immediate reward of zero as well as the next PR state (which corresponds to the subsequent PR review stage, denoted by the next in-line code comment of the same PR) within the step function. This process of action selection repeats, with the agent estimating the overall PR review outcome for each ensuing PR review stage, and receiving corresponding rewards of zero within the step function. This cyclical process continues until the agent has reviewed all the PR review stages within a PR, at which point the episode ends.

Upon the conclusion of the episode, the agent chooses an effective action, which is the final prediction for the entire PR after reviewing individual stages, and is granted a non-zero reward dependent on the accuracy of this prediction. Each episode represents a single PR from the training dataset, where the agent is rewarded zero after every step, and receives a non-zero reward at the conclusion of each episode. Upon the commencement of a new episode, the RL environment is reset via the reset function, and a new PR is introduced to the agent from the training data, thereby launching a new learning loop. The entire procedure is then iterated for the subsequent episode. The total count of timesteps for model training is preliminarily set at 600,000, which corresponds with the total number of PR comments in the dataset, ensuring the agent undergoes a thorough exposure to a wide variety of PR states throughout the training phase.

Throughout the training process, the RL model aims to discern an optimal course of action by maximizing the aggregate rewards received from its interactions with the environment. As each episode advances, the agent incrementally perfects its prediction policy based on the experienced rewards, endeavoring to attain the highest possible accuracy in estimating PR outcomes at every review stage for the samples present within the training dataset.
Algorithm 2: Training the RL-based PR Outcome Prediction Model

Input : Training dataset $TD = \{(pr_1, o_1), (pr_2, o_2), \ldots, (pr_i, o_i)\}$ /* PR State-outcome pairs, state is the current PR review stage */ /* where $pr_i = \{c_1, c_2, \ldots, c_n\}$ such that $c_n$ is the PR review stage for in-line code comment $n$ in PR no. $i$ from $TD$ */

Input : No. of PR samples in training data = $i$

Input : Total timesteps $T = 600,000$

Output: Trained RL-based PR prediction model

1. Initialize the RL environment
2. Set the action space $A = \{0, 1\}$ // Actions: 0 (reject), 1 (accept)
3. Set the observation space $O$ with the selected PR characteristics as described in $TD$ Set $step_t = 1$ // Number of timesteps
4. Set $E = 0$ // Number of Episodes
5. while $step_t < T$ do
6. Call reset function to reset the environment for a new episode
7. Set $t = 1$ // Step within the episode
8. Set initial state $s_1 = pr_1$
9. Set $n$ equal to the number of in-line code comments in $pr_1$
10. repeat
11. Select action $a_t$ using RL policy $\pi$: $a_t = \pi(s_t)$ ;
12. Perform action $a_t$ in the environment to get the next state $s_{t+1}$ and the immediate reward $r_t$ ;
13. Update the RL model using the observed reward $r_t$:
   UpdateModel($s_t, a_t, r_t, s_{t+1}$) ;
14. Set $s_t = s_{t+1}$ ;
15. Increment $t$: $t = t + 1$ ;
16. Increment $step_t$: $step_t = step_t + 1$ ;
17. until $t > n$;
18. Receive final reward based on the correctness of the final prediction and update the model.
19. Increment $E$: $E = E + 1$ ;
20. end while
21. Return the trained RL-based PR prediction model
For our experiments, we set the total timesteps as a variable, and conduct experiments to pinpoint the optimal number of timesteps for efficient model learning.

5.2.4 Training Parameters

In the preceding section, which outlined the training process for our predictive model, we underscored our preference for the DQN algorithm offered by the StableBaselines3 framework. However, it is worth noting that the preliminary results obtained from A2C and PPO algorithms are also included, although their performance was suboptimal. Despite this, we made a conscious decision to not proceed with these two algorithms for more comprehensive analysis. Furthermore, we refrained from attempting to experiment with other values for the trade-off factor or timestep value in an effort to optimize the parameters of these two algorithms.

The choice to fixate on the DQN algorithm for further analysis and experimentation is supported by its superior performance demonstrated in the initial findings, thus offering a solid basis for further exploration and optimization. This narrowed focus allows us to dedicate our resources and attention towards refining the DQN algorithm and obtaining a more accurate and efficient predictive model for PR outcomes with the PR Comments Dataset.

To initiate the training of the RL model, we employ the base classes of the pertinent RL algorithms in StableBaselines3, with the parameters set to their default values. During the training phase, the model undergoes training using the learn() method, which facilitates the learning process based on the interactions with the PR environment and the corresponding rewards derived from these interactions.

In evaluation mode, we assess the model’s performance by using the predict() method, making predictions based solely on knowledge acquired during training. This mode ensures no policy updates, allowing us to evaluate the model’s ability to generalize and accurately predict unseen data without altering the decision-making process.

It is worth noting that the DQN, A2C, and PPO algorithms adhere to the exact parameters as discussed and depicted in Section 4.2.4 “Training Parameters”. There is; however, one exception: the ‘timesteps’ parameter within the learn() method is set at 600,000 for the initial results.

With respect to the network architecture, for the DQN methodology, we employ the ‘MultiInputPolicy’ class, mirroring the approach taken in Chapter 4 which integrates the Rectified Linear Unit (ReLU) activation function. The CombinedExtractor serves as the feature extractor, designed to flatten the observation space, which is of the Dict type. The default optimizer enlisted for this purpose is Adam.

The structural design of the neural network starts with an input layer of dimension 8, correlating to the count of PR comment factors present in the training dataset. Following this, the first hidden layer manifests as a fully connected layer encompassing 8 input features and producing 64 output features (or neurons). Within the hidden layers, the ReLU activation function is applied, dictating the transformation of the weighted sum of inputs into output values for nodes within
the network layer.

The second hidden layer also materializes as a fully connected layer, which accepts the 64 output features from the preceding layer and creates 64 new output features. Lastly, the output layer, which receives the 64 output features from the former layer, anticipates the outcomes of PRs via a vector containing 2 elements as outlined in Listing 8.

Listing 8: Neural Network Architecture - DQN

```python
(q_net): Sequential(
    (0): Linear(in_features=8, out_features=64, bias=True)
    (1): ReLU()
    (2): Linear(in_features=64, out_features=64, bias=True)
    (3): ReLU()
    (4): Linear(in_features=64, out_features=2, bias=True)
)
```

Summary. Section 5.2.1 presents a novel approach to PR outcome prediction using just the PR discussions and RL. Section 5.2.2 defines the State Space, Action Space, and the Reward Function for effective PR predictions.

5.3 Comparative Analysis

As touched upon in Chapter 3, Section 3.4 on “Evaluation Metrics”, for the evaluation of this RL-based model, we have again opted for a collection of metrics including accuracy, F1-score, G-mean score, and AUC-ROC score. These measures serve to provide a comprehensive evaluation of the prediction techniques.

To evaluate the effectiveness of the RL-based PR outcome prediction model presented in Chapter 4, we referenced existing studies and selected a few notable techniques for comparative analysis. However, developing the second RL model was indeed a unique challenge, primarily due to the scarcity of relevant research work to draw upon. The model was constructed with the specific aim of predicting PR outcomes based on each individual review stage, using only in-line code comments as predictors. This is a unique approach that has not been extensively explored, leading to limited references in existing literature. While we found studies that employed sentiment analysis on GitHub developer comments, they did not align closely with our specific research goal. Most of these works focused on understanding the overall sentiment of developers’ discussions, their impact on collaboration, or project success. However, none of these studies used sentiment analysis in the context of predicting the outcomes of PRs.

The ability to predict this outcome based solely on the sentiment and content of in-line code comments opens up new avenues for understanding and optimizing the code review process. It offers a potentially powerful tool for project managers,
software engineers, and other stakeholders to anticipate bottlenecks and issues that could affect project timelines and quality.

Given this, and building on the techniques identified in Chapter 4, we chose to employ a Naive Bayes classifier as our baseline model, with Random Forest and XGBoost serving as our comparative benchmarks. In line with our approach in Chapter 4, we implement the Naive Bayes and Random Forest classifiers using the scikit-learn Python package, while for XGBoost, we employ the xgboost Python library.

Recognizing the imbalance present in our dataset, and to facilitate a comparative analysis of different techniques under varied resampling conditions, we also implement Random Under Sampling and Random Over Sampling, both of which are explained in detail in Section 3.5. Given the relatively high IR of approximately 0.26 in our training data, we use both RUS and ROS techniques to normalize the IR to 0.5, effectively balancing our dataset for better model performance.

5.3.1 Experiments

Primary Results

As we move forward with the discussion on the results derived from our experiments, this subsection focuses on presenting the primary results, all the while referencing the experiments and findings of Chapter 4 for context. Table 5.4, Table 5.5 and Table 5.6 summarize the performance of different RL algorithms—namely, A2C, PPO, and DQN—when applied to our second RL formulation that uses the PR Comments Dataset for training.

For these initial results, the dataset was partitioned into training and testing subsets, with an 80/20 split. The trade-off factor used in the cost-sensitive reward function for A2C and PPO was set at $k = 1 \times IR$, while for DQN, we chose $k = 2 \times IR$, where IR is the IR in the data. The decision to set a different trade-off value for DQN was informed by the results of Chapter 4.

With A2C and PPO, we observed an extreme bias towards predicting the majority class action, which in this context represents PR acceptance (value of 1 for the factor merged-or-not). For a trade-off factor of $k = 2 \times IR$, these models yielded a G-mean score of 0.0, indicating a complete failure to predict the minority class. To address this, we chose a smaller trade-off factor of $k = 1 \times IR$ for these two algorithms. However, this adjustment did not lead to any notable improvement in the A2C’s performance, as evidenced by the results in Table 5.4. PPO, on the other hand, showed a slight improvement, but its performance was still considerably poor.

Finally, one crucial parameter that varies during our primary results is the timesteps value used during the model’s learning phase. To begin with, we set the timesteps value to 600,000, which approximately equals the total number of inline code comments in our PR Comments Dataset. This value signifies the total number of PR review stages encompassed by all the PRs in our dataset. Our goal is to assess the model’s performance improvement when we systematically increase the timesteps value. Therefore, we increment it by 100,000 in each subsequent run, observing whether this increase facilitates an enhancement in the model’s predictive capabilities.
Table 5.4: Performance of A2C Algorithm for Outcome Prediction with PR Comments Dataset

<table>
<thead>
<tr>
<th>Total Timesteps (T)</th>
<th>G-mean</th>
<th>AUC-ROC</th>
</tr>
</thead>
<tbody>
<tr>
<td>T = 600,000</td>
<td>0.00</td>
<td>0.50</td>
</tr>
<tr>
<td>T = 700,000</td>
<td>0.02</td>
<td>0.50</td>
</tr>
<tr>
<td>T = 800,000</td>
<td>0.00</td>
<td>0.50</td>
</tr>
<tr>
<td>T = 900,000</td>
<td>0.00</td>
<td>0.50</td>
</tr>
<tr>
<td>T = 1,000,000</td>
<td>0.02</td>
<td>0.50</td>
</tr>
</tbody>
</table>

- 80/20 split: train.shape (471831, 15) and test.shape (116266, 15)
- Imbalance Ratio = 0.26 and trade-off factor (k) = 1*IR

Table 5.5: Performance of PPO Algorithm for Outcome Prediction with PR Comments Dataset

<table>
<thead>
<tr>
<th>Total Timesteps (T)</th>
<th>G-mean</th>
<th>AUC-ROC</th>
</tr>
</thead>
<tbody>
<tr>
<td>T = 1,000,000</td>
<td>0.41</td>
<td>0.52</td>
</tr>
<tr>
<td>T = 1,200,000</td>
<td>0.52</td>
<td>0.56</td>
</tr>
<tr>
<td>T = 1,400,000</td>
<td>0.44</td>
<td>0.51</td>
</tr>
<tr>
<td>T = 1,500,000</td>
<td>0.49</td>
<td>0.52</td>
</tr>
</tbody>
</table>

- 80/20 split: train.shape (471831, 15) and test.shape (116266, 15)
- Imbalance Ratio = 0.26 and trade-off factor (k) = 1*IR

The most noteworthy outcomes arise from the implementation of the DQN algorithm. For a trade-off factor of k = 2*IR, we initialize the timesteps value at 600,000. We then methodically test different values of timesteps, increasing this initial value by increments of 100,000, up to a maximum of 1,500,000. The top three performances achieved with this range of timesteps are consolidated and presented in Table 5.6.

Table 5.6: Performance of DQN Algorithm for Outcome Prediction with PR Comments Dataset

<table>
<thead>
<tr>
<th>Total Timesteps (T)</th>
<th>Accuracy (%)</th>
<th>F-1 Score</th>
<th>G-mean</th>
<th>AUC-ROC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Class-0</td>
<td>Class-1</td>
<td>Overall</td>
<td>Class-0</td>
</tr>
<tr>
<td>T = 800,000</td>
<td>83</td>
<td>94</td>
<td>92</td>
<td>0.81</td>
</tr>
<tr>
<td>T = 1,000,000</td>
<td>34</td>
<td>80</td>
<td>75</td>
<td>0.32</td>
</tr>
<tr>
<td>T = 1,500,000</td>
<td>52</td>
<td>45</td>
<td>49</td>
<td>0.29</td>
</tr>
</tbody>
</table>

- 80/20 split: train.shape (471831, 15) and test.shape (116266, 15)
- Imbalance Ratio = 0.26 and trade-off factor (k) = 2*IR

Our observations reveal that the most effective value for the timesteps parameter when utilizing the DQN algorithm with a reward function trade-off factor set at k = 2*IR is 800,000. To ensure the consistency and validity of our findings, we executed the model training three times, maintaining identical settings in each run. Despite this repetition, the average G-mean score remained stable at
approximately 0.78. Based on these consistent results, we chose to proceed with
the DQN algorithm for further analysis and comparisons, setting the timesteps
at 800,000 and the trade-off factor at $k = 2 \times IR$.

Summary. The optimum value of the total_timesteps ($T$) parameter for DQN
with $k = 2 \times IR$ for an 80/20 data split is $T = 800,000$ yielding a G-mean Score
of 0.88.

Effects of Data Resampling and Comparative Experiment
Analysis

In previous discussions, we identified Random Forest and XGBoost as compara-
tive techniques, and Naive Bayes as the baseline model. We divided the dataset
into an 80/20 split for training and testing purposes, respectively, and imple-
mented these three prediction techniques to assess their performance. Further-
more, we conducted Random Over Sampling and Random Under Sampling on the
training dataset, integrating these methods with Random Forest Classifier and
XGBoost models for further predictive analysis. The results of these operations
are compiled and summarized in Table 5.7.
Table 5.7: Comprehensive Comparative Performance Analysis of All Predictive Models

<table>
<thead>
<tr>
<th></th>
<th>RL-DQN</th>
<th>Baseline (Naive Bayes)</th>
<th>Random Forest Classifier</th>
<th>XGBoost Classifier</th>
<th>Random Forest Classifier</th>
<th>XGBoost Classifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall Accuracy (%)</td>
<td>92</td>
<td>77</td>
<td>78</td>
<td>79</td>
<td>76</td>
<td>74</td>
</tr>
<tr>
<td>F-1 Score</td>
<td>Class-0</td>
<td>0.81</td>
<td>0.06</td>
<td>0.11</td>
<td>0.05</td>
<td>0.15</td>
</tr>
<tr>
<td></td>
<td>Class-1</td>
<td>0.95</td>
<td>0.87</td>
<td>0.87</td>
<td>0.88</td>
<td>0.86</td>
</tr>
<tr>
<td>G-mean</td>
<td></td>
<td>0.88</td>
<td>0.18</td>
<td>0.25</td>
<td>0.17</td>
<td>0.31</td>
</tr>
<tr>
<td>AUC-ROC</td>
<td></td>
<td>0.89</td>
<td>0.50</td>
<td>0.52</td>
<td>0.51</td>
<td>0.52</td>
</tr>
</tbody>
</table>

80/20 split: train.shape (471831, 15) and test.shape (116266, 15) while Imbalance Ratio = 0.26

- For DQN, Imbalance Ratio = 0.26 and trade-off factor (k) = 2*IR, Total Timesteps (T) = 800,000
- For ROS, Imbalance Ratio = 0.5 and train.shape: (557065, 15)
- For RUS, Imbalance Ratio = 0.5 and train.shape: (301361, 15)
5.3.2 Contrasting the Two RL Approaches

In this study, two distinct RL formalizations were proposed to analyze the PR review process. Both predictive models utilize a Markov Decision Process, aiming to predict PR acceptance and rejection accurately.

Comprehensive Formalization of PR Characteristics

Dataset and Characteristics. Utilizes over 1.3 million PRs and includes 72 diverse characteristics to provide a broad perspective on the PR review outcome.

Performance. The RL-DQN model achieved a G-mean score of 0.83, outperforming comparative predictive techniques, particularly in predicting PR rejection.

PR Discussions-Based Formalization

Focus. Concentrates solely on PR discussions, specifically the in-line code comments associated with PRs.

Innovative Techniques. Includes sentiment analysis and comment-related feature extraction, along with optimizing the reward function and total timesteps parameter.

State-of-the-Art Performance. Achieved a new benchmark G-mean score of 0.88, far surpassing classical predictive methods and even the comprehensive formalization.

Comparative Insights

Emphasis on Developer Discussions. This study highlights the critical role of developer dialogues, feedback, and interactions in the PR review process. The results discussed in Section 5.3.1 provide substantial evidence to support this claim.

Superiority of the Discussions-Based Model. The RL-DQN model demonstrates exceptional ability in leveraging PR discussions, offering an unconventional approach to predicting PR outcomes, as highlighted in Table 5.7. It outperforms both the comprehensive formalization and classical techniques in PR outcome predictions.

These dual formalizations, focusing on a comprehensive view of PR characteristics and the unique angle of PR discussions, offer distinct insights into the PR review process. The performance of the PR discussions-based model establishes a novel benchmark for future studies, underscoring the importance of human interactions in the process. On the other hand, the comprehensive formalization still provides valuable insights, although not matching the performance of the discussions-based approach.
Summary. The PR discussion-based formalization to predict outcomes, surpasses the Comprehensive PR Characteristics formalization, setting a new benchmark G-mean Score of 0.88 with parameters $k = 2 \times IR$ and $T = 800,000$. This not only sets a new standard for future research but also suggests that discussion-based models may be more robust and applicable in real-world automated PR review systems.
Chapter 6

Understanding GitHub PRs: An Empirical Study on Review Processes and Merge Times through Developer Survey

As the concluding segment of our research, we conducted an exploratory survey targeting software developers and engineers. The underpinning objective was to delve deep into the developers’ perspectives regarding the PR review processes and the quality of these reviews. To gather insightful data, our carefully crafted survey was designed to delve into the methodologies employed by developers during PR reviews, discern the criteria they prioritize in evaluating a submitted PR, and pinpoint the factors they consider pivotal to the outcome and merge-time of PRs on GitHub.

This section can be encapsulated by its three-fold contribution:

1. **A Qualitative Insight.** We undertook a qualitative study engaging with professional software developers and engineers. This granted us a firsthand viewpoint into their perspective on code review quality.

2. **Decoding Developer Perception.** Our comprehensive survey analysis resolves the intricacies of how developers perceive the code review process. It pinpoints the various factors that come into play during the review and isolates the factors developers view as pivotal in determining the outcome and merge time of PRs.

3. **An Open Resource for Research.** In the spirit of transparency and advancing knowledge, we are providing access to a dataset consisting of 22 anonymized survey responses. This can serve as a foundation for further research and exploration in this domain.

The insights garnered from this survey were instrumental in making strategic decisions and design choices in our second RL formalization for predicting PR outcomes, as elaborated in Chapter 5. Specifically, survey results regarding the preferred locations for comments during the review process guided our decision to focus on in-line code comments for dataset curation. Moreover, feedback on the developers’ PR review approach emphasized the vital role of discussions in promoting collaboration and communication. These insights ultimately informed our decision to center the RL formalization solely on PR discussions, providing an innovative perspective for PR outcome prediction.

### 6.1 Survey Design

We designed a survey protocol following Carleton University’s guidelines for online research, adhering to the Tri-Council Policy Statement: Ethical Conduct for
After careful evaluation by Carleton University’s Research Ethics Boards, in alignment with TCPS2, we received approval on May 2, 2023 (Ethics Clearance ID # 119296), effective until May 31, 2023.

The survey was carefully structured into three distinct sections. The initial section delved into the participant’s demographic and professional background, featuring six primary questions, along with an optional seventh question. Prioritizing participant confidentiality, the survey was designed to safeguard anonymity. No sensitive details that might compromise participant identity were solicited, ensuring the survey remained entirely anonymous.

The subsequent section transitioned to a set of questions focused on PR factors and review practices. This section presented participants with two multiple-choice queries and a pair of questions grounded in the Likert-scale, enabling a structured feedback mechanism.

Concluding the survey, the third section was crafted to prompt more detailed insights from the participants. It comprised two open-ended questions, providing an avenue for respondents to further describe their PR review experiences and techniques. A tabulated breakdown, highlighting the nature and type of questions incorporated within the survey, is outlined in Table 6.1. The design of the developer survey followed standard practices as described in [58], ensuring a methodologically sound approach to data collection.

<table>
<thead>
<tr>
<th>Question Description</th>
<th>Type</th>
<th>Additional Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Demographic Information: Role, Tenure in Role, Years in Geographically Distributed Projects, Region of Residence, Average Monthly PRs Reviewed (Workload)</td>
<td>Options: i) GitHub Merge ii) Squashing iii) Cherry-picking</td>
<td>Options: Strongly Agree, Agree, Undecided, Disagree, Strongly Disagree</td>
</tr>
<tr>
<td>Select the type of merge that you use predominantly</td>
<td>Multiple Choice Question</td>
<td>Options: i) GitHub Merge ii) Squashing iii) Cherry-picking</td>
</tr>
<tr>
<td>Rate the factors that affect the outcome of the PR review process</td>
<td>Likert-scale</td>
<td>Strongly Agree, Agree, Undecided, Disagree, Strongly Disagree</td>
</tr>
<tr>
<td>Rate the factors that affect the merge time of the PR</td>
<td>Likert-scale</td>
<td>Strongly Agree, Agree, Undecided, Disagree, Strongly Disagree</td>
</tr>
<tr>
<td>During the review process, where do you most commonly provide the comments?</td>
<td>Multiple Choice Question</td>
<td>Options: i) On PR itself ii) On PR Code/In-line Code iii) In Individual Commits</td>
</tr>
<tr>
<td>What steps do you follow when asked to review a PR?</td>
<td>Open-ended Question</td>
<td></td>
</tr>
<tr>
<td>What factors do you use to examine the quality of the contributions?</td>
<td>Open-ended Question</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.1: Survey Questions and Response Types for GitHub PR Review Process Study

Participant Recruitment

The recruitment process for our survey participants relied on leveraging LinkedIn, a professional networking platform.* Given the specificity of our requirements, our recruitment was tailored towards software developers and engineers with a

---

1. https://tcps2core.ca/welcome
2. https://www.linkedin.com/
distinct proficiency in the GitHub environment and its pull-based development methodology.

To ensure the survey reached the appropriate demographic, we circulated both the survey and essential accompanying documents—such as invitational materials and consent forms—within developer-centric groups on LinkedIn. The baseline eligibility criteria mandated that participants should possess at least a year’s experience in software development, coupled with familiarity with GitHub’s modus operandi.

Using the analytical capabilities provided by LinkedIn, we observed that our survey invitation gained significant traction, registering over 800 impressions. Of this vast audience, approximately 65 qualified individuals expressed a keen intent to partake in the survey. Their enthusiasm manifested in diverse ways—ranging from direct private messages to comments, reactions on the post, and other inquiries.

The survey was live from 3rd May 2023 and concluded on 13th May 2023, spanning a total duration of 10 days. Recognizing the challenges that some faced in adhering to this timeframe, we extended the deadline for a select few; four participants, to be precise. This flexibility resulted in us receiving three additional responses beyond our initially stipulated window.

Ultimately, our coordinated efforts resulted in a total of 22 completed survey responses. While this might initially seem modest in volume, it is crucial to consider the context. When contrasted against the 65 genuinely interested participants, this translates to a commendable response rate of 34% (22/65). For perspective, this significantly surpasses the benchmarked minimum response rate of 10% as outlined in established literature [58].

Selecting Relevant PR Factors

In our survey, as reflected in the associated Table 6.1, we carefully crafted questions based on a combination of research insights and practical considerations. Here’s a breakdown:

1. **Likert-scale Questions on PR Factors.** These questions are influenced by three primary sources:

   - **Research Insights.** Our comprehensive literature review brought forward factors like ‘Technical Debt’, ‘Quality of PR Description’, ‘Project Age’, and ‘PR Reviewer Workload’ which have been shown to be impactful.

   - **Data-Driven Factors.** The PR dataset we used in Chapter 4’s RL formalization also guided our choices, ensuring our factors are grounded in real-world data.

   - **Additional Considerations.** We supplemented our list with additional factors that we believed would provide valuable insights. These include the ‘PR Merge Type Used’, ‘Type of the Project’, the ‘Project Domain’, and the ‘Project License’ which could be commercial, public-domain, or even unlicensed.
2. **Multiple-choice Question on PR Merge Types.** We aimed to understand the preferred method developers use when merging PRs. Our options were [2]:

- **GitHub Merge.** This is the most straightforward option, using the merge button provided in the GitHub interface.
- **Cherry-picking Merge.** In this method, a developer selects specific commits from a PR and integrates them into the main repository, without making any changes.
- **Squashing.** Here, a developer compiles all commits from a PR into a single new commit. They can then make further changes if necessary before adding this consolidated commit to the main repository.

3. **Understanding Comment Preferences during PR Reviews.** Comments play a critical role in PR reviews. On GitHub, these comments can be categorized mainly into three types [3]:

- **PR-level Comments.** General comments made on the PR itself.
- **In-line Code Comments.** Specific feedback or remarks on particular lines within the PR.
- **Commit Comments.** Comments directly associated with specific commits within the PR.

We included a multiple-choice question in our survey to understand which type of comments developers predominantly use during their review process. This categorization helps in understanding the granularity and focus of feedback during PR reviews.

### 6.2 Survey Response Analysis Framework

#### Demographic Questions

To better understand the backgrounds of our participants, we produced several visual representations. By examining the responses in Section 1, which delves into the demographic background and work habits of the participants, we gained insights into various aspects. This section touched upon details like the participants’ professional experience, their specific roles in the software industry, and the typical workload they handle. These visualizations served as a comprehensive overview, shedding light on the diverse profiles and practices of our survey respondents.

#### Likert-scale Questions

For Section 2 of the survey, we aimed to discern developers’ perceptions of the influence of different PR factors on PR outcomes and merge times using two Likert-scale questions. These questions were framed using a 5-point Likert scale, ranging from “Strongly Agree” to “Strongly Disagree”, with the intent of measuring the perceived significance of each PR factor. A selection of “Strongly Agree”

for a PR factor implies the respondent perceives it as having a strong influence on the PR outcome or merge time. Conversely, a selection of “Strongly Disagree” conveys the opposite. To quantify these perceptions, we adopted a method to compute the weighted average for each PR factor. For the Likert-scale responses in our survey, each option is associated with a numerical weight to represent the strength of agreement or disagreement:

- **Strongly Agree**: \(w_5\)
- **Agree**: \(w_4\)
- **Undecided**: \(w_3\)
- **Disagree**: \(w_2\)
- **Strongly Disagree**: \(w_1\)

Where, typically, \(w_5 = 5, w_4 = 4, w_3 = 3, w_2 = 2,\) and \(w_1 = 1.\) Let the number of responses corresponding to each option be represented as:

- **Strongly Agree**: \(r_5\)
- **Agree**: \(r_4\)
- **Undecided**: \(r_3\)
- **Disagree**: \(r_2\)
- **Strongly Disagree**: \(r_1\)

The weighted average for any PR factor, \(X,\) is given by:

\[
\text{weighted\_avg}_X = \frac{w_5 \times r_5 + w_4 \times r_4 + w_3 \times r_3 + w_2 \times r_2 + w_1 \times r_1}{R}
\]

Where \(R\) is the total number of responses, such that \(R = r_5 + r_4 + r_3 + r_2 + r_1.\)

The representative response for PR factor \(X\) is the Likert-scale option whose weight is closest to \(\text{weighted\_avg}_X.\)

We visualized these weighted averages for a comprehensive overview, and a detailed explanation of the results follows in the subsequent section.

**Open-ended Questions**

In the third section of our survey, which comprised of two open-ended questions, we employed the “grounded theory” methodology [59] to examine and understand the qualitative data we collected. Grounded theory is an investigative approach typically used in social sciences to build theory from data inductively. Here’s a more in-depth look into our method:

1. **What is Coding in Qualitative Research?** In qualitative research, coding is a foundational step that entails carefully analyzing raw data to identify recurring themes, patterns, or categories. These themes offer deeper insights into the participant’s perspectives, beliefs, or experiences.

2. **Initial Reading and Open Coding.** Our analysis began with an “open coding” process. During this phase, we immersed ourselves in the data by reading each participant’s response several times. As we did this, we labeled individual ideas, insights, or perspectives with distinct codes, which could be a word or a brief phrase that captures the essence of that particular segment of the response.
3. Axial Coding - Finding Connections. Once the open coding process was complete, we transitioned to “axial coding”. In this phase, we examined the relationships between the individual codes. Codes with shared or overlapping meanings were grouped together, leading to the formation of broader themes or categories. This step allows us to reduce redundancy and organize our findings into more palatable chunks.

4. Theme Finalization and Summarization. With our themes clearly outlined, we then delved deeper into each theme to truly understand the nuances and intricacies they encapsulated. Finally, a comprehensive summary of each theme was crafted, offering a broader view of the collective thoughts and feelings of the participants regarding the open-ended questions.

This approach ensures that our findings remain firmly grounded in the participants’ actual responses, giving an authentic account of their experiences and perspectives.

6.3 Results

6.3.1 Demographic and Workload Analysis

In designing our survey, a paramount consideration was maintaining the privacy and anonymity of our respondents. This measure was taken to ensure that participants felt secure, reducing any hesitancy they might have in providing honest and detailed responses. Specifically, we gathered information on:

1. **Professional Role.** Understanding the exact role of the participant helps in gauging the perspective from which they might evaluate a PR. For instance, a senior developer might have different views on PRs compared to a junior developer.

2. **Experience in Role.** The number of years a participant has spent in their current role provides a measure for their level of expertise and the depth of their understanding in their specific domain.

3. **Experience in Geographically Distributed Projects.** By this, we refer to projects wherein PR contributors and integrators are located in different countries. This measure helps us understand the challenges and dynamics a participant might face when collaborating with colleagues from diverse cultural and geographic backgrounds. Different time zones, communication styles, and collaboration methods can all play into how PRs are evaluated and integrated.

4. **Region.** While this might seem like a simple geographical marker, it can also give us indirect insights into the technological infrastructure and prevalent corporate cultures that the respondent might have been exposed to.

5. **Average Workload (PRs per Month).** This data point provides a measure of the volume of work each participant handles in the context of PR evaluations.
Geographical and Professional Distribution of Participants

Our survey drew participants from various corners of the world, reflecting a diverse set of backgrounds and experiences. A quick breakdown reveals:

1. **Asia.** Standing out as the dominant contributor, we had 13 participants from Asia region.

2. **North America.** This region had a total of 9 representatives in the survey.

Given the researcher’s recent relocation from India to Canada and the primary use of LinkedIn for participant recruitment, it is logical to see a significant number of responses originating from both Asia and North America. Shifting our focus to the professional roles of our participants, an interesting pattern emerges:

1. **Variety in Job Titles.** Among the 22 participants, there was a diverse range of roles, summing up to 13 distinct job titles, highlighting the diverse specializations within the software development field.

2. **Classification of Roles.** Given our primary objective was to target software developers and engineers, we undertook a manual review of the job titles. Post review, we found:

   - **Software Developer.** Four participants were aligned closely with this specific designation.
   - **Software Engineer.** The roles of 18 participants fell under the expansive category of ‘Software Engineer’, reflecting the overarching and encompassing nature of the software engineering domain.

Analyzing Developers’ Workload and Experience Patterns

To assess developers’ workloads, we initially employed a straightforward scatter chart to represent each participant’s workload visually, as shown in Fig. 6.1. Although our data set comprised of a modest 22 responses, we observed an anomaly with one participant handling an average of 100 PRs monthly.

To better understand this data distribution, we superimposed three critical reference lines on our scatter plot, which plotted PR Workload against Participants. The first line represented the mean workload, while the other two lines were set at one standard deviation above and below this average. In most scenarios that fit a normal distribution, the majority of data points will fall within this range, making any data points outside of it, potential outliers.

To further understand the correlation between a developer’s experience and their PR workload, we embarked on a more granular analysis as shown in Fig. 6.2. This examination involved:

1. **Defining Parameters for Comparison.** We chose two key metrics:
   - The tenure participants had in their current roles.
   - Their exposure to geographically distributed projects.
2. **Visualization through Scatter Plot.** We charted these parameters against the PR workload, intentionally excluding the earlier identified outlier to ensure the data’s clarity.

3. **Role Differentiation.** To make the data more interpretable, points on the scatter plot were color-coded based on the participant’s broader role designation - be it a Software Engineer or a Software Developer.

4. **Observations.** The resulting visualization did not present any obvious trends. This lack of pattern reinforced the idea that individual PR workloads can vary widely, uninfluenced by their professional experience or involvement in distributed projects.

To refine our analysis, we clustered participants based on their years of experience and undertook the following methodological steps:

1. **Segmentation Based on Experience.** We segmented our participants into distinct groups based on their years of experience, using two-year intervals.

2. **Average Workload Computation.** For each of these experience segments, we computed the average number of PRs handled per month.

3. **Visualization with a Bar Chart.** We chose a bar chart to represent these averages for each experience segment. It is worth noting that to maintain data integrity, the outlier participant, with a workload of 100 PRs/month, was left out. Bar chart in Fig. 6.3

4. **Key Insights.** Those with 0-2 years of experience had the heaviest average workload of 22 PRs per month. Those in the subsequent bracket of 2-4 years

---

Figure 6.1: Scatter Plot of Participants vs. Mean Monthly PR Workload with Average and Outlier Reference Lines
had a close average of 20 PRs. Participants with 4-6 years of experience saw a reduction with an average of 15 PRs per month. While the most experienced group, with 10-12 years, reported the lowest workload, averaging a mere 5 PRs monthly. It is worth noting that only one respondent fell into the 10-12 year experience bracket. Therefore, caution should be exercised when drawing definitive conclusions based on this particular data point.

6.3.2 Developer Preferences for PR Merges and Review Interaction

In the pull-based development approach, there’s a heightened degree of flexibility when handling PRs. They can choose to integrate a minimal portion of it, beneficial to the project, or opt to merge the entire PR. When merging the entire PR, they have another choice to make: either retain the original commits from the PR, which serves to maintain a rich historical record, or they can ‘squash’ these commits into a singular commit. The latter method ensures a streamlined and tidy commit history on the master branch.

Understanding developers’ preferences for these merging techniques can offer insights into their workflow priorities, be it preserving historical data or maintaining a neat commit history. Consequently, as discussed in the prior sections, we integrated a multiple-choice question into our survey to discern the predominant merge strategies employed by developers. As illustrated in the accompanying pie chart in Fig. 6.4 among the 22 respondents, 13 predominantly used the built-in GitHub Merge for their PRs. Eight favored the Squashing method, while Cherry-picking was the least popular, chosen by just one participant.

Comments are integral to the PR review process, serving as a primary channel for feedback, suggestions, and fostering discussions between developers. It is pivotal to understand where developers are most inclined to place their comments,
as this can offer insight on the preferred areas of engagement and interaction during PR reviews.

To gauge this, we incorporated a multiple-choice question into our survey, inquiring about the participants’ common sites for commenting during the PR evaluation phase. The resulting pie chart in Fig. 6.5 illuminates a dominant preference: a remarkable 77.3% of respondents primarily commented on the PR code directly, utilizing in-line code comments. Meanwhile, 22.7% preferred commenting on the PR itself. Interestingly, none of the participants opted for commenting within individual commits, indicating that this approach might be less favored or less known among our surveyed group.

### 6.3.3 Perceptions of Influential Factors for PR Outcome

In the pursuit to understand the key factors that influence the outcome of a PR, we utilized a Likert-scale question. This allowed us to delve into developers’ perceptions by asking them to rate their agreement with various factors. By assigning values to the five scales (Strongly Agree, Agree, Undecided, Disagree, Strongly Disagree), we calculated the weighted averages for each factor to uncover the predominant sentiment.

The graph in Fig. 6.6 illustrated some standout factors deemed significant for PR outcome, including Test Inclusion (weighted average of 4.36), PR Reviewer Experience (weighted average of 4.27), Technical Debt (weighted average of 4.18), PR Size in Lines of Code (LOC) (weighted average of 4.14), Quality of PR Description (weighted average of 4.09), and CI Build Status (weighted average of 4.09). These factors all resonated with an effective response of ‘Agree.’

Notably, none of the effective responses aligned with Disagree, Strongly Disagree, or Strongly Agree. Thus, participants’ views seemed to polarize around the ‘Agree’ and ‘Undecided’ camps.

To gain a more nuanced perspective, we contrasted the distribution of re-
Figure 6.4: Predominant PR Merge Types Selected by Participants

Figure 6.5: Participants’ Preferred Locations for Providing Comments During the PR Review Process (No Responses for ‘In Individual Commits’ Option)

responses across the Strongly Agree, Agree, Disagree, and Strongly Disagree scales as shown in Fig. [6.7]. This analysis revealed that for factors like Test Inclusion and Quality of PR Description, respondents predominantly selected Strongly Agree or Agree. Meanwhile, factors like PR Size, PR Reviewer Experience, Source Churn, and Technical Debt had minimal dissent, with only one or two responses choosing Disagree. Upon a closer look at the full range of responses, including those categorized under the Undecided scale, it became apparent that most of the previously mentioned factors had 2 or fewer responses marked as Undecided. This was in contrast to the factors of Source Churn and Quality of PR Description, where each had 4 responses falling into the Undecided category. This divergence suggests a higher level of ambiguity or varying perceptions regarding the influence of Source Churn in comparison to other factors assessed.

Consequently, the findings emphasize the perceived importance of factors like Test Inclusion, Quality of PR Description, PR Size, PR Reviewer Experience, and
Technical Debt in influencing PR review outcomes. These insights reinforce the multifaceted nature of PR assessment and the need for comprehensive evaluation criteria that consider both quantitative and qualitative metrics.

### 6.3.4 Perceptions of Influential Factors for PR Merge Time

In our exploration of the factors perceived to influence PR merge time, we applied a method analogous to our approach for identifying the factors influential to PR review outcomes. Utilizing a 5-Likert-scale, we assigned values to the five distinct scales and calculated a weighted average for each surveyed factor to observe an effective response.

The chart in Fig. 6.8 showcases that PR Reviewer Workload emerged as a paramount factor, registering a weighted average of 4.5 and consequently denoting a Strongly Agree response. Other factors gaining prominence include PR Size (weighted average of 4.27), Test Inclusion (weighted average of 4.05), and PR Reviewer Experience (weighted average of 4.05), all of which yielded an effective response of Agree.

It is noteworthy to mention that none of the surveyed features produced an effective response of Disagree or Strongly Disagree, leading to the majority of responses leaning toward either Undecided or Agree, aside from PR Reviewer Workload which commanded a Strongly Agree stance.

Delving further, we analyzed the response distribution across the Strongly Agree, Agree, Disagree, and Strongly Disagree scales as highlighted in Fig. 6.9. The analysis revealed that both PR Reviewer Workload and PR Size received unanimous agreement, with no Disagree or Strongly Disagree responses. Test Inclusion and CI Build Status followed, each having one response as Disagree. Surprisingly, PR Reviewer Experience had one response as Strongly Disagree.

However, when we extended our analysis to include the Undecided scale, a more nuanced picture emerged. PR Size and Test Inclusion each gathered 2 responses marked Undecided, whereas CI Build Status accumulated 4 such responses. These findings, when taken as a whole, allow us to conclude that PR Reviewer Workload, PR Size, PR Reviewer Experience, and Test Inclusion are
Figure 6.7: Distribution of Participant Responses on Influential Factors to PR Outcome: A Contrast Across Strongly Agree, Agree, Disagree, and Strongly Disagree Scales

indeed perceived by participants as influential factors for PR merge time. The data reflects an intricate relationship, with general alignment in Agree but varying degrees of uncertainty as seen in the Undecided responses, highlighting the multifaceted nature of these factors in determining PR merge time.

**Summary.** Developer’s Perception of Most Influential factors

- For PR Outcome: Test Inclusion, PR Reviewer Experience, Technical Debt, PR Size in LOC, Quality of PR Description, and CI Build Status
- For PR Merge time: PR Reviewer Workload, PR Size in LOC, Test Inclusion, and PR Reviewer Experience
Figure 6.8: PR Merge Time Factors: Effective Response Analysis

Figure 6.9: Distribution of Participant Responses on Influential Factors to PR Merge Time: A Contrast Across Strongly Agree, Agree, Disagree, and Strongly Disagree Scales
6.3.5 Analysis and Thematic Interpretation of the PR Review Process

For the open-ended question regarding the procedures used by the participants when reviewing PRs, we first performed a manual examination of individual responses to correct any spelling errors. Three changes were made: ‘coment’ was altered to ‘comment’ in two instances, and ‘reviewr’ was corrected to ‘reviewer’ in one instance. Following these preliminary corrections, we initiated our manual coding methodology to assess and condense the responses.

Applying the grounded theory method to the collected responses, we analyzed the data in three distinct stages, culminating in the identification of overarching themes as below which are described in detail in Table 6.2:

Open Coding. We identify individual ideas, insights, or perspectives and label them with specific codes.

Axial Coding. We examine the relationships between the individual codes and group them together into broader themes or categories.

Theme Finalization. Delve deeper into each theme to understand the nuances fully. The final themes described below:

1. Understanding and Analyzing Changes. Participants in the review process prioritize understanding the proposed changes within a PR. They diligently read the PR’s title and description and may focus exclusively on file changes to grasp the technical details. This assessment goes beyond surface-level examination, incorporating an evaluation of the feature or ticket description to confirm alignment with project requirements, scope, and objectives. The reviewers are intent on thoroughly dissecting the changes to ascertain how well they conform to the project’s goals and standard coding practices. This initial step fosters a comprehensive understanding, setting the stage for a thorough and informed review of the PR.

2. PR Documentation Standards and Conventions. Reviewers emphasize the importance of adhering to particular commit conventions within the PR process, as this helps maintain uniformity throughout the codebase. The supporting documentation, encompassing comments and supplementary materials, is expected to abide by specific standards and guidelines. Such careful adherence ensures that the details are communicated clearly and are accessible to all project stakeholders.

3. Collaboration and Communication. The responses highlighted the importance of teamwork in the PR review process. Participants emphasized the role of clear decision-making and open dialogue, using in-line comments and attachments (such as files, images, or test results) to pinpoint specific concerns or offer evidence-based feedback. This approach reflects a collective effort, highlighting the importance of shared understanding and constructive communication in ensuring code alignment with project standards and objectives.

4. Coding Conventions and Organization. This theme captures the attention and importance given by the participants to the structuring and
standardization of code during the PR review process. This involves adherence to predefined coding conventions, such as naming variables consistently and following specific style guidelines. Participants stressed the need to avoid technical debt by ensuring that the coding style aligns with the agreed-upon standards and that the overall code organization is coherent and maintainable. The focus on these aspects during the PR review process ensures that the code is both functional and comprehensible, thereby facilitating future developments.

5. **Code Quality Assurance, Optimization and Reusability.** This theme encapsulates various steps taken by the participants that they consider vital to the PR review process. It underscores the importance of examining code for design, complexity, and efficiency in the PR review process. Participants stressed the need for optimization, adherence to principles like “Don’t Repeat Yourself” (DRY), and ensuring code is reusable and maintainable. These steps reflect a commitment to quality, promoting code that meets immediate needs while facilitating long-term maintainability.

6. **External Resources Evaluation.** This theme focuses on the careful analysis of external tools, libraries, and tests in a PR. Participants stressed the importance of evaluating these resources for compatibility, performance, and alignment with project standards during the PR review process. This step ensures the selected external components fit the project’s needs without introducing unforeseen complexities or compromising quality.

7. **Testing and Verification.** This theme encapsulates the thorough process undertaken by participants to ensure that the changes proposed in a PR are valid and secure. This involves a comprehensive review of the components, test results, and security checks. Participants expressed the necessity to verify all changes, including ensuring that both the build and tests have passed successfully. The emphasis on testing and verification procedures reflects a commitment to maintaining high standards of code quality, functionality, and security in the project.

8. **Review Process Formalization.** This theme highlights the participants’ use of a structured approach in reviewing PRs. Utilizing an in-built review process, they ensure consistent evaluations according to predefined criteria. This formalization emphasizes the importance of specific review strategies aligned with project goals, fostering a more systematic and efficient evaluation of the proposed changes. This theme specifically pertains to the responses that reference the existence of an in-built review process, without elaborating on further details.
Table 6.2: Analysis of Steps Taken in PR Reviews: Initial Codes, Axial Codes, and Final Themes (Based on 22 Responses)

<table>
<thead>
<tr>
<th>Initial Codes</th>
<th>Axial Coding (Grouped Codes)</th>
<th>Final Themes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read PR title/description</td>
<td>PR Documentation</td>
<td>Understanding and Analyzing Changes</td>
</tr>
<tr>
<td>Check feature/ticket description</td>
<td>Understanding Requirements</td>
<td>PR Documentation Standards and Conventions</td>
</tr>
<tr>
<td>Review file changes only</td>
<td>Understanding Scope and Objective</td>
<td></td>
</tr>
<tr>
<td>Commit conventions</td>
<td>Accept/reject changes</td>
<td>Feedback and Decision Making</td>
</tr>
<tr>
<td>Comment on code</td>
<td>In-line comments</td>
<td>Comments and Communication</td>
</tr>
<tr>
<td>Attach files, images in PR, test results</td>
<td></td>
<td>Collaboration and Communication</td>
</tr>
<tr>
<td>Coding Conventions</td>
<td>Variable naming conventions</td>
<td>Coding Standards</td>
</tr>
<tr>
<td>Technical Debt</td>
<td>Ensure coding style and guidelines</td>
<td>Code Structuring</td>
</tr>
<tr>
<td>Check code design</td>
<td>Coding Conventions and Organization</td>
<td></td>
</tr>
<tr>
<td>Check complexity</td>
<td>Check for optimization</td>
<td>Complexity Analysis</td>
</tr>
<tr>
<td>Check if unnecessary code</td>
<td>Naming and Design Consistency</td>
<td></td>
</tr>
<tr>
<td>Check code quality</td>
<td>Code Quality Checks</td>
<td>Code Quality Assurance, Optimization and Reusability</td>
</tr>
<tr>
<td>DR\Y principle</td>
<td>Code Efficiency and Optimization</td>
<td></td>
</tr>
<tr>
<td>Code reusability</td>
<td>Code Maintainability and Reusability</td>
<td></td>
</tr>
<tr>
<td>Code maintainability</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Checking tests, libraries used</td>
<td>Review components, test results</td>
<td>Libraries and Tools Evaluation</td>
</tr>
<tr>
<td>Security checks</td>
<td>Security checks</td>
<td>External Resources Evaluation</td>
</tr>
<tr>
<td>Test changes</td>
<td>Test changes</td>
<td>Testing and Verification</td>
</tr>
<tr>
<td>Verify all changes</td>
<td></td>
<td>Build and Test Validation</td>
</tr>
<tr>
<td>Check build and tests passed</td>
<td></td>
<td>Security and Compliance</td>
</tr>
<tr>
<td>In-built review process</td>
<td>Specific Review Strategies</td>
<td>Review Process Formalization</td>
</tr>
</tbody>
</table>

Summary. The study of the PR review process, as described by participants, illustrates a detailed and methodical approach. It emphasizes a comprehensive understanding of changes, strict adherence to coding and documentation conventions, collaboration, quality assurance, resource evaluation, rigorous testing, and systematic review protocols. These aspects collectively contribute to maintaining code quality, ensuring project alignment, and promoting a cooperative environment within software development.

6.3.6 Analysis and Thematic Interpretation of Factors to Ascertain Quality of PRs

For the open-ended question on the factors used by the participants to examine the quality of PRs, in the first step, we manually checked individual responses for any spelling mistakes and made one change: ‘readablity’ changed to ‘readability’ (found in one response). After the initial corrections, we began with our manual coding approach to analyze and summarize the responses.

Applying the grounded theory method to the collected responses, we analyzed the data in three distinct stages, as below and described in Table 6.3:

**Open Coding.** We identify individual ideas, insights, or perspectives and label them with specific codes.

**Axial Coding.** We examine the relationships between the individual codes and group them together into broader themes or categories.
Theme Finalization. Delve deeper into each theme to understand the nuances fully. Final themes as follows:

1. Coding Conventions and Readability. This theme emphasizes the adherence to established coding conventions and the importance of code readability in the PR review process. Participants stressed that following standard conventions, such as consistent naming and styling, is integral to maintaining a coherent codebase. Additionally, ensuring that code is readable allows for a shared understanding among team members and facilitates future maintenance. Together, these aspects underline a commitment to quality and collaboration, promoting well-organized and accessible code in software development projects.

2. Code Quality and Optimization. Participants in the survey prioritize both the quality and efficiency of code when evaluating PRs. They carefully scrutinize code correctness through various testing methods, ensuring it is free from errors and aligns with project goals. Performance optimization is also essential; participants look for the use of library functions, adherence to best practices, and avoidance of unnecessary complexity to enhance code efficiency. This theme underscores a balance between writing precise, functional code and optimizing it for performance.

3. Code Reusability and Maintainability. The theme of code reusability and maintainability emerged as a significant factor in evaluating the quality of PRs among survey participants. They emphasized the importance of writing code that can be used across different parts of the project or in future projects, adhering to principles that promote reusability. Alongside this, participants stressed the need for maintainable code, meaning it must be structured in a way that allows future developers to understand and modify it easily.

4. Contributor Role and Experience: This theme highlights the nuanced way participants assess the quality of PRs by examining the contributor’s role, experience level, and historical contributions. Recognizing the significant influence of the individual behind the PR, reviewers often weigh the experience, past performance, and the software development engineer (SDE) level of the contributor. Trust is also built over time with consistent, high-quality contributions, further influencing the assessment. This integrated approach goes beyond technical evaluation, signifying the belief that the quality of a PR is not solely based on the code’s characteristics, but also deeply influenced by the contributor or the PR author.

5. PR Documentation, Size and Comment Conventions. This theme underlines the significance of clear documentation, comments, and adherence to proper size in the PR review process. Participants value detailed explanations and inline comments that clarify complex code, along with the size of the PR, as they often reflect underlying considerations. These elements together ensure understanding, standardization, and future maintainability, playing a crucial role in evaluating the quality of PRs.
6. **Testing and Verification.** This theme emphasizes the systematic process of ensuring that the proposed changes within a PR are accurate, secure, and efficient. Participants in the study highlighted the need to conduct thorough testing, including unit tests, corner cases, and handling of potential errors. The presence of security measures, clear examples of input and output, and a well-structured approach to the problem statement further contribute to this process. By scrutinizing these aspects, reviewers are able to confirm the reliability and robustness of the code, underscoring the commitment to maintaining high standards of quality and integrity in the project.

<table>
<thead>
<tr>
<th>Initial Codes</th>
<th>Axial Coding (Grouped Codes)</th>
<th>Final Themes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Clarity and Style</td>
<td>Code Readability and Clarity</td>
<td>Coding Conventions and Readability</td>
</tr>
<tr>
<td>Coding Conventions</td>
<td>Code Standards and Formatting</td>
<td></td>
</tr>
<tr>
<td>Code Readability</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code Testing</td>
<td>Code Quality and Correctness</td>
<td></td>
</tr>
<tr>
<td>Code Correctness</td>
<td>Code Efficiency and Optimization</td>
<td></td>
</tr>
<tr>
<td>Performance Optimization</td>
<td>Code Efficiency through Libraries</td>
<td></td>
</tr>
<tr>
<td>Utilization of Library Functions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code Complexity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code Efficiency</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code Maintainability</td>
<td>Code Reusability and Maintainability</td>
<td>Code Reusability and Maintainability</td>
</tr>
<tr>
<td>Reusability</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trust in Contributors</td>
<td>Trust and Contribution Frequency</td>
<td>Contributor Role and Experience</td>
</tr>
<tr>
<td>PR Author Contribution Frequency</td>
<td>SDE Level Consideration</td>
<td></td>
</tr>
<tr>
<td>PR Author Experience</td>
<td>Experience and Contributor History</td>
<td></td>
</tr>
<tr>
<td>SDE Level Consideration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Documentation</td>
<td>PR Documentation &amp; Comments</td>
<td>PR Documentation, Size and</td>
</tr>
<tr>
<td>Comments &amp; Unit Test</td>
<td>PR Size and Composition</td>
<td>Comment Conventions</td>
</tr>
<tr>
<td>Code Comments</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lines of Code</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Security Measures</td>
<td>Security Compliance</td>
<td></td>
</tr>
<tr>
<td>Test Results</td>
<td>Unit Testing</td>
<td></td>
</tr>
<tr>
<td>Corner Test Cases</td>
<td>Problem-solving and Error Handling</td>
<td>Testing and Verification</td>
</tr>
<tr>
<td>Error Handling</td>
<td>Code Testing and Examples</td>
<td></td>
</tr>
<tr>
<td>Approach to Problem Statement</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Presence of Sample Input/Output</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6.3: Analysis of Factors Used to Examine PR Quality: Initial Codes, Axial Codes, and Final Themes (Based on 22 Responses)

**Summary.** The study examining the quality of PRs revealed six key themes that emphasize a balance between technical proficiency, individual contributor’s role, and collaboration. Participants value coding standards, efficiency, reusability, trust in the contributor, clear documentation, and rigorous testing. These findings collectively illustrate the intricate method employed by software professionals in PR reviews, emphasizing a concentration on factors that ensure coding proficiency, teamwork, and individual credibility.
Chapter 7

Conclusion and Future Work

This chapter serves as a comprehensive recapitulation of the entire thesis, bringing together the diverse elements that have been explored and examined in the preceding chapters. The chapter also outlines future directions for the work, identifying potential areas for further exploration or refinement. This helps to place the research within a broader context, and indicates its relevance and potential impact going forward. Further, the chapter acknowledges and details the threats to validity. This honest examination of limitations provides transparency and acknowledges the potential weaknesses of the study.

7.1 Conclusion

This thesis has undertaken a multifaceted exploration in the domain of software development, employing Reinforcement Learning (RL) to predict Pull Request (PR) outcomes on GitHub and seeking to understand the strategies and influential factors employed during the PR review process.

**RL Formalizations: Holistic Perspective of PR Characteristics**

By systematically formalizing the decision-making process in PR review as an RL problem and using a comprehensive dataset of over 1.3 million PRs with 72 characteristics, the first formalization demonstrated RL’s potential in predicting PR outcomes and established a framework for further exploration. The DQN algorithm excelled in adaptability and accuracy, with G-mean scores of 0.83, 0.79, and 0.72 in 80/20, 50/50, and 20/80 data splits, respectively. In the special test cases, it achieved high accuracies of 86% and 87% for PR rejection predictions in 50/50 and 20/80 splits and 73% and 60% for PR acceptance in the same splits. In comparison, the XGBoost Classifier (used after ROS with an 80/20 split) attained a G-mean of 0.77 but only 5% accuracy for PR rejection outcome in special test cases. Additionally, the exploration of actions, states, episodes, and reward functions provided further insights into optimal choices for effective PR outcome predictions.

**RL Formalizations: Utilizing PR Discussions**

The second formalization, focusing exclusively on PR discussions and in-line code comments, presented an innovative approach to PR outcome predictions using a dataset of around 0.6 million comments from 66,281 PRs. The DQN algorithm’s ability to predict outcomes at each stage of the review process, achieving a G-mean of 0.88, marks a significant advancement, establishing a baseline for future studies. Among competitors, Random Forest Classifier (used after RUS with an 80/20 split) came the closest with a G-mean of 0.35. This approach underscores...
the dynamic nature of PR reviews and offers a novel perspective that complements
the more comprehensive view of the first formalization.

**Comparative Analysis of RL Agent Performance**
The comparative analysis of both RL formalizations with existing PR outcome prediction methodologies and traditional classifiers like Random Forest Classifier and XGBoost Classifier showcases the strengths of the DQN models. Its superiority across various data splits, resampling techniques, and specialized test cases emphasizes its robustness, adaptability, and potential as a valuable tool for real-world applications. Particularly, the models’ effectiveness at predicting PR rejection sets a new benchmark for future studies.

**Insights from Developer/Software Engineer Survey**
Through an exploratory survey, the thesis uncovers detailed insights into the PR review process:

1. **Steps in Review Process.** Understanding and analyzing changes by reading PRs; adhering to documentation standards and following commit conventions; fostering communication through comments and feedback; maintaining coding conventions; assuring code quality and efficiency; evaluating external resources such as libraries; conducting testing and verification; aligning with predefined criteria for assessment (if applicable).

2. **Quality Assessment Criteria.** Including coding conventions and readability; code quality, optimization, reusability, and maintainability; contributor role and experience; PR documentation conventions; testing and verification.

3. **Key Influential Factors for PR Outcome.** Test Inclusion, PR Reviewer Experience, Technical Debt, PR Size in LOC, Quality of PR Description, and CI Build Status.

4. **Key Influential Factors for PR Merge Time.** PR Reviewer Workload, PR Size in LOC, Test Inclusion, and PR Reviewer Experience.

In conclusion, the dual perspective offered by the two distinct RL formalizations, together with the insights from the exploratory developer survey and a comprehensive literature review, brings forward a rich tapestry of understanding and innovation. While one formalization provides a broad, comprehensive view, the other focuses on the granularity of PR discussions, offering step-wise predictions. Both perspectives, in unity, offer new directions for research, development, and real-world applications.

The DQN algorithm’s notable performance in predicting PR rejection outcomes was achieved through optimizing the trade-off factor and a cost-sensitive reward function. This novel approach to addressing data imbalance emphasizes the depth and innovation of the research. In summary, this thesis stands as a pioneering work, advancing the understanding of predictive modeling, collaboration, and innovative use of RL in PR predictions. By publishing a curated PR Comments dataset and the Survey Response dataset\(^1\), it not only offers valuable insights for the present but also lays a solid foundation for future exploration and research. The source code for all the models is available on GitHub\(^2\).

\(^1\)https://zenodo.org/record/8271704
\(^2\)https://github.com/kernish/RL-PullRequest-Predictions
7.2 Future Research

This research aimed to comprehensively explore RL in PR review predictions, detailing the network structures, training parameters, and comparison techniques. Specialized test cases demonstrated the models' robustness and efficiency. We further enhanced sensitivity and accuracy in predicting PR outcomes by implementing a cost-sensitive reward function with a trade-off parameter, optimizing to achieve favorable results.

However, this research has unexplored avenues that hold potential for future work:

1. **Parameter Optimization.** Our efforts to optimize parameters offered in the StableBaselines3 framework using the popular hyperparameter optimization framework Optuna[^1] were hindered by computational constraints. Even with state-of-the-art computing clusters, with dedicated Graphics Processing Unit (GPU) and allocated memory in the range of 500 GB, we struggled to complete optimization within 21 days. The clusters that we used had a time limit of 21 days after which any running scripts are automatically terminated. Future studies could focus solely on parameter optimization, taking into account the problem-specific nature of parameter tuning and the different RL formalizations, to further enhance the model’s efficacy.

2. **Addressing Data Imbalance.** While the integration of resampling techniques like Synthetic Minority Over-Sampling (SMOTE) [49] and Adaptive Synthetic Sampling (ADASYN) [60] into the RL formalizations was initially considered, it was determined to be beyond the scope of the current research. Future work could explore employing RL techniques to generate synthetic data for resampling imbalanced datasets. Such an endeavor would constitute a separate study, requiring extensive investigation and analysis, and could further enhance the existing methodologies.

3. **Expanding Action Space and Perspectives.** The thesis presents RL formalizations from two angles: one emphasizing a broad view of PRs with an extensive dataset and the other targeting the intricacy of PR discussions. Given the complexity of the datasets, the action space was limited to two discrete actions (acceptance or rejection). Future work might merge these perspectives and explore an expanded action space, allowing more nuanced actions such as engaging in discussions, marking for changes, or reviewing the state. These actions could be guided by insights from this thesis and could foster innovative approaches to PR review automation using RL agents.

4. **Reviewer Recommendations.** Building upon the comprehensive state space design that encompasses a wide array of PR-related features, as detailed in Chapter 4, future work could modify the existing RL formalization to create an agent capable of making targeted recommendations to reviewers. Such an agent could guide reviewers to focus on specific aspects or parts of a PR that may require closer scrutiny, thereby enhancing the review process.

[^1]: https://optuna.org/
5. Specialized Datasets. The introduction of more complex actions implies the necessity for specialized datasets encapsulating all relevant information for such innovation. A targeted approach could involve focusing on a specific GitHub repository known for its active development and large contributor base. This could pave the way for RL agents that emulate human review processes more closely.

In summary, while this research represents a significant stride in applying RL to PR review predictions, there are promising directions for future exploration. These include thorough optimization of all training parameters, integration of multiple perspectives and a richer action space, and the cultivation of specialized datasets. These pathways can potentially lead to more sophisticated, human-like PR review automation using RL, contributing to the evolving landscape of software development.

7.3 Threats to Validity

The foundation of our work rests upon a decade of research on pull-based development, from which we extract features pertinent to PR decision-making, consequently offering novel perspectives for outcome predictions. While this approach builds upon significant previous research, taking advantage of established ideas, it also leads us to inherit some of their existing limitations. Below, we enumerate these limitations and their potential implications:

1. Data Preprocessing Limitations. In the process of data preprocessing, we opted to remove specific factors and consequently, a number of PRs to eliminate redundancy or excise missing values as outlined in Chapter 4, Section 4.1. The consequences of these decisions on the representativeness of the dataset used in the predictive RL-based models remain uncertain. This uncertainty may constitute a potential threat to the validity of our conclusions regarding the generalizability of model performance.

2. Data Bias Concerns. Despite having amassed extensive datasets for both RL formalizations and employing various data splits for training and testing, a complete eradication of data bias in building models across different contexts remains elusive. Although the diversity within our data and the rigorous test cases lend some degree of generalizability, it cannot fully mitigate the intrinsic biases that may be present in the models.

3. Platform-Specific Results. Our data collection was exclusively derived from the GitHub platform, raising questions about the applicability of our findings to other social coding platforms such as Gerrit and GitLab to name a few. The variance in factors influencing PR decisions across different platforms, reflecting their unique operational modes and social interactions, might limit the external validity of our results.

4. Sentiment Analysis Limitations. In our second RL formalization, we utilized the VADER framework for sentiment analysis, focusing on in-line comments to predict PR outcomes. While we used individual emotion scores to enhance generalizability and reduce bias, the inherent limitations
of VADER are inevitably transferred to our study. This may have implications on the preciseness of sentiment classification within our predictive models.

5. **Hyperparameter Tuning.** We designed a cost-sensitive reward function with a trade-off factor to manage data imbalance, an approach that, along with total timesteps optimization, led to new benchmark performances. However, the absence of a comprehensive hyperparameter optimization stage in our research may present limitations in achieving optimal performance.

6. **Survey Design and Analysis Bias.** While we took great care in designing an unbiased developer survey, the manual coding of open-ended questions and the analysis of other question types (e.g., Likert-scale, Multiple Choice, Demographic questions) may have introduced bias. Despite adhering to a strict protocol as detailed in Chapter 6, Section 6.2, some degree of bias might remain in the results. Moreover, the relatively small sample size of 22 responses may limit the generalizability of insights drawn from the survey.

In conclusion, these limitations must be recognized as integral aspects of our study, influencing our interpretations and the potential applications of our findings. While we have sought to minimize their impact through rigorous design and analytical procedures, future work must consider these constraints in both the replication and expansion of our research. By acknowledging these limitations, we provide a clear path for subsequent studies to build upon our work, refining methodologies, and extending our understanding of RL-based models in PR decision-making.
REFERENCES


[33] A. Kumar, M. Khare, and S. Tiwari, “Sentiment analysis of developers’ comments on GitHub repository: A study,” in 2022 14th International Conference on Advanced Computational Intelligence (ICACI), 2022, pp. 91–98.


[53] V. Kumar, G. S. Lalotra, P. Sasaki$\text{l}$a, D. S. Rajput, R. Kahir$\text{u}$, K. Lak$\text{v}$shmann$\text{a}$, M. Shorfuzzaman, A. Alsufy$\text{a}$ni, and M. Uddin,


## APPENDICES

### A Additional Tables

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>policy</td>
<td>the policy model to use, here obs_space is type Dict</td>
<td>'MultiInputPolicy'</td>
</tr>
<tr>
<td>env</td>
<td>the environment to learn from</td>
<td>instance of class prEnv</td>
</tr>
<tr>
<td>learning_rate</td>
<td>the learning rate, it can be from 1 to 0</td>
<td>0.00001</td>
</tr>
<tr>
<td>buffer_size</td>
<td>size of the replay buffer</td>
<td>1000000</td>
</tr>
<tr>
<td>learning_starts</td>
<td>how many steps of the model to collect transitions for before</td>
<td>50000</td>
</tr>
<tr>
<td>batch_size</td>
<td>minibatch size for each gradient update</td>
<td>32</td>
</tr>
<tr>
<td>tau</td>
<td>the soft update coefficient between 0 and 1</td>
<td>1</td>
</tr>
<tr>
<td>gamma</td>
<td>the discount factor</td>
<td>0.99</td>
</tr>
<tr>
<td>train_freq</td>
<td>update the model every train_freq steps</td>
<td>4</td>
</tr>
<tr>
<td>gradient_steps</td>
<td>how many gradient steps to do after each rollout</td>
<td>1</td>
</tr>
<tr>
<td>replay_buffer_class</td>
<td>replay buffer class to use. If None, it will be automatically selected</td>
<td>None</td>
</tr>
<tr>
<td>replay_buffer_kwargs</td>
<td>keyword arguments to pass to the replay buffer on creation</td>
<td>None</td>
</tr>
<tr>
<td>optimize_memory_usage</td>
<td>enable memory efficient variant of the replay buffer at a cost of</td>
<td>FALSE</td>
</tr>
<tr>
<td>target_update_interval</td>
<td>update target network every target_update_interval environment steps</td>
<td>100000</td>
</tr>
<tr>
<td>exploration_fraction</td>
<td>fraction of entire training period over which the exploration rate is reduced</td>
<td>0.1</td>
</tr>
<tr>
<td>exploration_initial_eps</td>
<td>initial value of random action probability</td>
<td>1</td>
</tr>
<tr>
<td>exploration_final_eps</td>
<td>final value of random action probability</td>
<td>0.05</td>
</tr>
<tr>
<td>max_grad_norm</td>
<td>the maximum value for the gradient clipping</td>
<td>10</td>
</tr>
<tr>
<td>stats_window_size</td>
<td>window size for the rollout logging, specifying the number of episodes to average the reported success rate, mean episode length, and mean reward over</td>
<td>100</td>
</tr>
<tr>
<td>tensorboard_log</td>
<td>the log location for tensorboard</td>
<td>log_path</td>
</tr>
<tr>
<td>policy_kwargs</td>
<td>additional arguments to be passed to the policy on creation</td>
<td>None</td>
</tr>
<tr>
<td>verbose</td>
<td>verbosity level, 0 for no output, 1 for info messages, 2 for debug messages</td>
<td>0</td>
</tr>
<tr>
<td>seed</td>
<td>seed for the pseudo random generators</td>
<td>None</td>
</tr>
<tr>
<td>device</td>
<td>device such as CPU, CUDA, etc. on which the code should be run</td>
<td>'auto'</td>
</tr>
<tr>
<td>_init_setup_model</td>
<td>whether or not to build the network at the creation of the instance</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

Table A.1: DQN Algorithm Training Parameters and Values
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>policy</td>
<td>the policy model to use. here obs_space is type Dict</td>
<td>'MultiInputPolicy'</td>
</tr>
<tr>
<td>env</td>
<td>the environment to learn from</td>
<td>instance of class prEnv</td>
</tr>
<tr>
<td>learning_rate</td>
<td>the learning rate, it can be from 1 to 0</td>
<td>0.0003</td>
</tr>
<tr>
<td>n_steps</td>
<td>the number of steps to run for each environment per update</td>
<td>2048</td>
</tr>
<tr>
<td>batch_size</td>
<td>minibatch size for each gradient update</td>
<td>64</td>
</tr>
<tr>
<td>n_epochs</td>
<td>number of epoch when optimizing the surrogate loss</td>
<td>10</td>
</tr>
<tr>
<td>gamma</td>
<td>the discount factor</td>
<td>0.99</td>
</tr>
<tr>
<td>gae_lambda</td>
<td>factor for trade-off of bias vs variance for Generalized Advantage Estimator</td>
<td>0.95</td>
</tr>
<tr>
<td>clip_range</td>
<td>clipping parameter, it can be from 1 to 0</td>
<td>0.2</td>
</tr>
<tr>
<td>clip_range_vf</td>
<td>clipping parameter for the value function, it can be from 1 to 0.</td>
<td>None</td>
</tr>
<tr>
<td>normalize_advantage</td>
<td>whether to normalize or not the advantage</td>
<td>TRUE</td>
</tr>
<tr>
<td>ent_coef</td>
<td>entropy coefficient for the loss calculation</td>
<td>0</td>
</tr>
<tr>
<td>vf_coef</td>
<td>value function coefficient for the loss calculation</td>
<td>0.5</td>
</tr>
<tr>
<td>max_grad_norm</td>
<td>the maximum value for the gradient clipping</td>
<td>0.5</td>
</tr>
<tr>
<td>use_sde</td>
<td>whether to use generalized State Dependent Exploration (gSDE) instead of action noise exploration</td>
<td>FALSE</td>
</tr>
<tr>
<td>sed_sample_freq</td>
<td>sample a new noise matrix every n steps when using gSDE</td>
<td>-1</td>
</tr>
<tr>
<td>target_kl</td>
<td>limit the KL divergence between updates, because the clipping is not enough to prevent large update</td>
<td>None</td>
</tr>
<tr>
<td>stats_window_size</td>
<td>episodes to average the reported success rate, mean episode length, and mean reward over</td>
<td>100</td>
</tr>
<tr>
<td>tensorboard_log</td>
<td>the log location for tensorboard</td>
<td>log_path</td>
</tr>
<tr>
<td>policy_kwargs</td>
<td>additional arguments to be passed to the policy on creation</td>
<td>None</td>
</tr>
<tr>
<td>verbose</td>
<td>verbosity level, 0 for no output, 1 for info messages, 2 for debug messages</td>
<td>0</td>
</tr>
<tr>
<td>seed</td>
<td>seed for the pseudo random generators</td>
<td>None</td>
</tr>
<tr>
<td>device</td>
<td>device such as CPU, CUDA, etc. on which the code should be run</td>
<td>'auto'</td>
</tr>
<tr>
<td>_init_setup_model</td>
<td>whether or not to build the network at the creation of the instance</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

Table A.2: PPO Algorithm Training Parameters and Values
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>policy</td>
<td>the policy model to use, here obs_space is type Dict</td>
<td>'MultiInputPolicy'</td>
</tr>
<tr>
<td>env</td>
<td>the environment to learn from</td>
<td>instance of class prEnv</td>
</tr>
<tr>
<td>learning_rate</td>
<td>the learning rate, it can be from 1 to 0</td>
<td>0.0007</td>
</tr>
<tr>
<td>n_steps</td>
<td>the number of steps to run for each environment per update</td>
<td>5</td>
</tr>
<tr>
<td>gamma</td>
<td>the discount factor</td>
<td>0.99</td>
</tr>
<tr>
<td>gae_lambda</td>
<td>factor for trade-off of bias vs variance for Generalized Advantage Estimator</td>
<td>2</td>
</tr>
<tr>
<td>ent_coef</td>
<td>entropy coefficient for the loss calculation</td>
<td>0</td>
</tr>
<tr>
<td>vf_coef</td>
<td>value function coefficient for the loss calculation</td>
<td>0.5</td>
</tr>
<tr>
<td>max_grad_norm</td>
<td>the maximum value for the gradient clipping</td>
<td>0.5</td>
</tr>
<tr>
<td>rms_prop_eps</td>
<td>RMSProp epsilon. It stabilizes square root computation in denominator of RMSProp update</td>
<td>1.00E-05</td>
</tr>
<tr>
<td>use_rmsprop</td>
<td>whether to use RMSProp (default) or Adam as optimizer</td>
<td>TRUE</td>
</tr>
<tr>
<td>use_sde</td>
<td>whether to use generalized State Dependent Exploration (gSDE)</td>
<td>FALSE</td>
</tr>
<tr>
<td>sed_sample_freq</td>
<td>sample a new noise matrix every n steps when using gSDE</td>
<td>-1</td>
</tr>
<tr>
<td>normalize_advantage</td>
<td>whether to normalize or not the advantage</td>
<td>FALSE</td>
</tr>
<tr>
<td>stats_window_size</td>
<td>window size for the rollout logging, specifying the number of episodes to average the reported success rate, mean episode length, and mean reward over</td>
<td>100</td>
</tr>
<tr>
<td>tensorboard_log</td>
<td>the log location for tensorboard</td>
<td>log_path</td>
</tr>
<tr>
<td>policy_kwargs</td>
<td>additional arguments to be passed to the policy on creation</td>
<td>None</td>
</tr>
<tr>
<td>verbose</td>
<td>verbosity level, 0 for no output, 1 for info messages, 2 for debug messages</td>
<td>0</td>
</tr>
<tr>
<td>seed</td>
<td>seed for the pseudo random generators</td>
<td>None</td>
</tr>
<tr>
<td>device</td>
<td>device such as CPU, CUDA, etc. on which the code should be run</td>
<td>'auto'</td>
</tr>
<tr>
<td>init_setup_model</td>
<td>whether or not to build the network at the creation of the instance</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

Table A.3: A2C Algorithm Training Parameters and Values
<table>
<thead>
<tr>
<th>Sub-section</th>
<th>Authors</th>
<th>Approach</th>
<th>Data</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Georgios Gousios et al. (2014)</td>
<td>Quantitative</td>
<td>GHTorrent data corpus</td>
<td>Used classification algorithms to identify influential factors: Random Forests, Logistic Regression, Naive Bayes, Support Vector Machines, Decision Trees, Adaboost with Decision Trees</td>
</tr>
</tbody>
</table>
|             | Oleksii Kononenko et al. (2018) | Quantitative and Qualitative | 1) PRs from Active Merchant’s GitHub repository  
2) Survey of 16 developers | Multiple Linear Regression Model to study PR review time  
Logistic Regression Model to study PR outcome |
|             | Xunhui Zhang et al. (2021) | Quantitative and Qualitative | 1) GHTorrent data corpus  
2) Systematic Literature Review | Used a Mixed-Effect Logistic Regression Model to explain the influential factors |
|             | Gunmar Kudrjavets et al. (2022) | Quantitative    | Public data from OSS projects on Gerrit and Phabricator               | 1) Used time-to-first-response, time-to-accept, time-to-merge for analysis.  
2) Used descriptive statistics, decision tree to analyze factors affecting review time |
|             | Oleksii Kononenko et al. (2016) | Qualitative     | Survey of 88 developers from Mozilla project on GitHub                | Identified experienced developers using the Bugzilla issue tracking system  
and used grounded theory with manual coding to analyze the survey data |
|             | Olga Baysal et al. (2016) | Quantitative and Qualitative | 1) Systematic Literature Review  
2) Code review data from Bugzilla issue tracking system for WebKit project  
3) Code review data from Chromium repository for Blink project | 1) Used Mann-Whitney U test to perform pairwise comparisons with significance threshold p < 0.05  
2) Used null hypothesis testing, analysis of variance, post-hoc testing, and Spearman’s correlation to ensure statistically significant associations |
|             | Danielle Moreira et al. (2015) | Quantitative    | GHTorrent data corpus                                                | Used Apron algorithm in WEKA tool to derive association rules |
|             | Panthip Pooput et al. (2018) | Quantitative    | Data extracted from the Ansible OSS projects’ GitHub repository      | Used Apron algorithm in WEKA tool to derive association rules |
|             | Valentina Lenarduzzi et al. (2019) | Quantitative    | 28 OSS Java projects on GitHub                                        | 1) Examined 4.5 million code quality flaws in 36,344 PRs  
2) Statistical analysis with contingency matrices, the X2 test, logistic regression, and ensemble classifiers  
3) Manually inspected 10% of both accepted and non-accepted PRs |
|             | Marcelino Campos et al. (2016) | Quantitative    | PR data from 6 OSS projects on GitHub                                 | Manually categorized pull-requests for analysis: design, documentation, test, build, project convention, performance, security debt |
|             | Jason Tay et al., (2014) | Quantitative    | 659,501 PRs across 12,482 GitHub projects                             | Applied a multi-level mixed effects logistic regression model to predict the likelihood of pull request acceptance |
|             | Rahul Iyer et al., (2021) | Quantitative    | 16,935 developer comments from 1,860 GitHub projects                  | 1) Extracted the big five personality traits (Openness, Extraversion, Agreeableness, Conscientiousness, and Neuroticism)  
2) Used mixed effects logistic regression to study the effects of personality traits  
2) First round of survey: 32 respondents (7% response rate)  
3) Final round of survey: 760 responses (18% response rate) |
|             | Georgios Gousios et al., (2016) | Quantitative and Qualitative | 1) GHTorrent data corpus  
2) First round of survey: 21 respondents (8% response rate)  
3) Final round of survey: 749 respondents (23% response rate) |
|             | Georgios Gousios et al., (2015) | Quantitative and Qualitative | 1) GHTorrent data corpus  
2) First round of survey: 21 respondents (8% response rate)  
3) Final round of survey: 749 respondents (23% response rate) |
|             | Jason Tay et al., (2014) | Quantitative    | 1) Interviewed 47 GitHub users  
2) Randomly selected 20 PRs from their previous study to collect 423 comments from 115 developers | Utilized a grounded theory method to assess contributions, identifying interaction categories among participants, and examining core and peripheral developers’ discussions to resolve issues |
|             | Di Chen et al. (2019) | Quantitative and Qualitative | 1) GHTorrent data corpus  
2) Survey through Crowdsourcing | Crowdsourcing study involving:  
1) 27 employers found on MTurk platform  
2) 77 hours of crowd time  
3) Used decision tree classifier to predict acceptance |

Table A.4: Summary of Literature Review: Authors, Years, Datasets, and Methodological Remarks (Part-1)
<table>
<thead>
<tr>
<th>Sub-section</th>
<th>Authors</th>
<th>Approach</th>
<th>Data</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>2) Code review data extracted form CRITIQUE, a web-based code review</td>
<td>2) Successfully interviewed 12 employees</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>tool used at Google</td>
<td>3) Survey of 44 respondents (45% response rate)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3) Code review data extracted from CRITIQUE, a web-based code review</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>tool used at Google</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Jiaxin Zhu et al. (2016)</td>
<td>Quantitative</td>
<td>1) 8 patch-based OSS projects and 4 pull-based OSS projects</td>
<td>1) Evaluated contribution effectiveness through three aspects: contribution effort, time interval, and contribution activeness</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2) A Specific Rails project that migrated from patch-based to GitHub</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Oleksii Kononenko et al. (2015)</td>
<td>Quantitative</td>
<td>1) Code review data extracted from Bugzilla system</td>
<td>2) And used significance tests along with descriptive statistics</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2) Code review data extracted from Mozilla-central repository</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Klissomara Dias et al. (2020)</td>
<td>Quantitative</td>
<td>Analyzed 73,504 merge scenarios from 100 Ruby and 25 Python projects</td>
<td>1) Applied SZZ algorithm to identify bug-inducing modifications</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>hosted on GitHub</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Emitza Guzman et al. (2014)</td>
<td>Quantitative</td>
<td>Focused on 29 projects with over 200 comments, analyzing a total of</td>
<td>2) Constructed a Multiple Linear Regression model to analyze the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>60,425 commit comments</td>
<td>correlation between code review quality and factors, using:</td>
</tr>
<tr>
<td></td>
<td>Aman Kumar et al. (2022)</td>
<td>Quantitative</td>
<td>135,000 issue comments, 95,000 PR comments, 23,000 commit comments</td>
<td>- Response variable: code review quality (buggy or not)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>of top 10 Python projects on GitHub</td>
<td>- Exploratory variables: technical, personal, and participation metrics</td>
</tr>
<tr>
<td></td>
<td>Daniel Pletea et al. (2014)</td>
<td>Quantitative</td>
<td>60,658 commit comments and 54,892 PR comments from GHTorrent Data</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mohammad Maseudur Rahman et al. (2017)</td>
<td>Quantitative</td>
<td>1,116 code review comments from 4 commercial systems on GitHub</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>within an anonymized company</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tapajit Dey et al. (2020)</td>
<td>Quantitative</td>
<td>50 PR-related factors from the GitHub repository of 4218 popular</td>
<td>1) Proposed a tool, RevHelper, that differentiates between useful and</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>NPM packages resulting in 483,988 PRs</td>
<td>non-useful code review comments</td>
</tr>
<tr>
<td></td>
<td>Jiang Jing et al. (2020)</td>
<td>Quantitative</td>
<td>Extracted 28 GitHub projects, consisting of a total of 221,096 PRs,</td>
<td>2) RevHelper uses Random Forest as the underlying technique</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>of which 154,851 were accepted PRs</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Caglas Evren Gerede et al. (2018)</td>
<td>Quantitative</td>
<td>From the Android project on Geritt, collected code review data for</td>
<td>1) Identified 7 influential features</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>11,633 CRs made between October 2008 and January 2012</td>
<td>2) Trained predictive models using Support Vector Machine, AdaBoost, Bagging, and Random Forest with the WEKA tool; Random Forest performed best.</td>
</tr>
</tbody>
</table>

Table A.5: Summary of Literature Review: Authors, Years, Datasets, and Methodological Remarks (Part-2)
B Additional Figures

Figure B.1: Visualizing the Phi Correlation Coefficient: Correlation Between Dichotomous Variables in Chapter 4
Figure B.2: Visualizing the Point-Biserial Correlation Coefficient: Correlation Between Continuous and Dichotomous Variables in Chapter 4
Figure B.3: Visualizing the Point-Biserial Correlation Coefficient: Correlation Between Continuous and Dichotomous Variables in Chapter 5
Figure B.4: Visualizing the Phi Correlation Coefficient: Correlation Between Dichotomous Variables in Chapter 5
<table>
<thead>
<tr>
<th>PR Factor</th>
<th>Strongly Agree</th>
<th>Agree</th>
<th>Undecided</th>
<th>Disagree</th>
<th>Strongly Disagree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bug Fix</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CI build status</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CI tools</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Friday Effect</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No. of comments PR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No. of commits</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No. of contributors to project</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No. of files changed</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No. of participants in comments</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PR author affiliation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PR author experience</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PR reviewer experience</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PR size (in LOC)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project age</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project Domain</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project License</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quality of Description of PR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Source Churn</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Technical debt</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test inclusion</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Type of project</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure B.5: Factor’s Influence on PR Outcome: Survey Response Distribution and Perceived Importance
Figure B.6: Factor’s Influence on PR Merge Time: Survey Response Distribution and Perceived Importance