PReview: A Benchmark Dataset for Pull Request Outcomes and Quality Analysis

Anonymous ACL submission

Abstract

In this work, we present a novel dataset specifically designed for predicting pull request (PR) outcomes using large language models (LLMs). Our dataset is the first to integrate textual and code-related features, allowing the use of LLMs in PR outcome prediction, in contrast to earlier techniques that rely on numerical datasets. To construct this dataset we collected and carefully filtered pull request data from six well-known repositories on GitHub, the largest platform for collaborative code development. The dataset consists of 300 pull requests (PRs), each labeled with 'green' and 'red' flags to predict whether the PR will be merged or rejected. The PRs are annotated based on key features such as PR title, body, comments, contributor statistics, code changes, and related issues. The merged-to-unmerged PR ratio in the dataset is approximately 2:1. To promote reproducibility and foster further research, we will publicly release the dataset. This work lays the groundwork for building intelligent systems that can assist in PR review and decision-making by leveraging the capabilities of LLMs.

1 Introduction

Pull requests (PRs) are a cornerstone of collaborative software development. They provide a structured mechanism for contributors to propose changes and for maintainers to review, discuss, and ultimately decide on merging them into the main codebase (Gousios et al., 2014). This workflow is vital for ensuring code quality, maintainability, and project coherence. But in busy projects with lots of activity, it can also become time-consuming and mentally exhausting for maintainers who have to handle a large number of PRs. Automating aspects of this decision-making process, such as predicting the likelihood of PR acceptance, has the potential to significantly reduce review bottlenecks (Maddila et al., 2023), streamline development cycles, and improve contributor experience.

Recent studies have applied machine learning (ML) to predict PR outcomes, using structured features like lines changed, response time, and contributor history (Azeem et al., 2020; Banyongrakkul, 2022; Chen et al., 2025; Zhao et al., 2019). Models ranging from logistic regression to deep learning and GNNs (Banyongrakkul and Phoomvuthisarn, 2023; Gupta, 2018) show moderate success but often overlook the textual and semantic context critical to human decisions (Ford et al., 2019; Lenarduzzi et al., 2021; Zhang et al., 2023). For example, XGBoost may learn that PRs with over 10 changed files often get rejected, but it can't tell that a simple typo fix in many files is low-risk. It treats PRs as flat feature rows, not rich, connected documents.

In contrast, large language models (LLMs) like GPT-4, Claude, CodeLlama, and DeepSeek-Coder are built to reason over unstructured and semi-structured text (Brown et al., 2020). They can synthesize information from titles, descriptions, commit messages, code changes, and comments—offering a more holistic view of PRs (Fang et al., 2022; Liu et al., 2019; Sakib et al., 2024). However, their effectiveness in software engineering tasks depends on the availability of datasets suited to their strengths. Most existing PR datasets are designed for traditional ML models (Chen et al., 2025), focusing on shallow metadata and lacking the rich language, code, and social context LLMs need.

To bridge this gap, we present a novel dataset of 300 annotated PRs from six popular GitHub repositories. Each PR includes titles, descriptions, commits, diffs, comments, contributor stats, and linked issues. We also provide manual annotations—green and red flags—capturing key indicators behind acceptance or rejection. This dataset supports research on PR reasoning with LLMs and enables development of tools that prioritize reviews, assist maintainers, and enhance code review efficiency.

Our study uses this dataset to explore two key research questions about LLMs:

RQ1: Can LLMs accurately predict pull request outcomes (merge or rejection) using the rich, annotated PR data?

RQ2: How effectively can LLMs identify and explain the specific factors, represented by green and red flags, that influence PR acceptance decisions within the full PR context?

2 Dataset

2.1 Repository Selection

We selected six popular GitHub repositories—transformers, rails, mui, eslint, flask, and react-hook-form—to ensure coverage across diverse languages and development styles. Selection was based on having at least 2,000 closed PRs to support meaningful sampling, frequent use of issue-linking syntax (e.g., `closes #123`) to enable PR–issue pairing, evidence of ongoing maintenance activity, and diversity in programming languages including Python, Ruby, JavaScript, and TypeScript. This strategy provided a balanced representation of frontend and backend tools, libraries, and frameworks with varying review practices.

2.2 Data Collection Pipeline

We built a two-stage pipeline using GitHub's GraphQL and REST APIs:

Stage 1: PR–Issue Pairing. We used the GraphQL field–`closingIssuesReferences` to extract PRs that formally reference issues, following the approach in (Işık et al., 2025). To avoid false matches, we validated each issue via REST API to ensure it originated from the same repository and existed at the time of PR creation.

Stage 2: Artifact Retrieval. For each valid PR–issue pair, we retrieved a comprehensive set of artifacts: PR metadata (including title, body, merge status, and lines changed), full code diffs and commit messages, reviewer feedback (both inline and top-level comments), the title and description of the linked issue, and contributor statistics such as prior PR history and acceptance rate. All retrieved data was saved in structured CSV format and subsequently imported into Google Sheets for annotation.

2.3 Sampling Strategy

From the broad set of verified PR–issue pairs extracted in Phase 1, we chose a smaller set of **300 examples** as the basis of downstream annotation and analysis. This selection aimed to ensure diversity in PR outcomes and characteristics of contributions while making it feasible for manual annotation. The final sample contains a combination of unmerged and merged pull requests, spanning a range of contribution types and complexity. These samples contain the appropriate coverage to accommodate the detection of green and red flags as well as outcome reasoning. Table 1 shows the final distribution of samples across all our selected repositories.

Repository	**Total**	**Merged**	**Unmerged**
huggingface/transformers	73	58	15
rails/rails	42	23	19
pallets/flask	52	32	20
eslint/eslint	52	37	15
react-hook-form/react-hook-form	29	24	5
mui/material-ui	52	32	20

Table 1: Distribution of PR–issue pairs selected from six repositories.

2.4 Flag-Based Annotation Framework

Each pull request was annotated using a dual-label framework consisting of **10 green** and **14 red** flags, representing positive and negative quality indicators. Green flags denote alignment with linked issues, test coverage, contributor reputation, clarity in communication, and responsiveness. Red flags highlight issues like test failures, poor or over-engineered code, scope mismatch, large diffs, or unaddressed reviewer concerns. Several of these quality indicators were adapted from or inspired by prior research on PR characteristics and acceptance criteria (Işık et al., 2025; Zhang et al., 2023; Lenarduzzi et al., 2021).

Category	**Green Flag**	**Red Flag**
Clarity	Clear title/desc	Vague/missing info
Code Size	Small ($\leq$300 lines)	Large (>500 lines)
Testing	Good tests, all pass	No or failing tests
Issue Link	Linked to issue	Unlinked/unrelated
Reviews	Positive, responsive	Negative or silent
Contributor Stats	Merge rate $\geq$80%	Merge rate <50%
Quality	Clean, efficient	Buggy or breaking

Table 2: Annotation criteria used for labeling.

Impact Level (Weight)	Green Flags	Red Flags
Critical (19)	–	Failing tests, Risks breaking compatibility, Incorrect or inefficient solution, Over-engineered solution
High (3)	Clear alignment with issue, Includes test cases, High contributor acceptance rate	PR not aligned with issue, Missing or inadequate test cases, Low contributor acceptance rate
Medium (2)	Positive sentiment in comments, Manageable number of changed lines, Responsive to feedback	Negative sentiment in comments, Large number of changed lines, Unresponsive to feedback
Low (1)	Clear PR title, description, and commit messages, Good code quality	Unclear or missing PR title, description and commit messages, Poor code quality

Table 3: Impact levels and corresponding green and red quality flags used during manual PR annotation.

To reflect their relative importance, flags were assigned impact weights: **low (1)**, **medium (2)**, **high (3)**, and **critical (19)**. Green flags contribute positively to a PR's score, while red flags contribute negatively. The critical tier was calibrated to outweigh all green signals (which total 18). Furthermore, we included a computed total score column, where the weights for each PR's green and red flags were added together. This was used as a sanity check to ensure consistency in the annotations. Table 3 lists the full flag set with their weights.

2.5 Annotation Protocol and Validation

All 300 examples were labeled collaboratively by the three authors using a dropdown-based interface with multi-select flags and an optional notes field for justifying edge cases. To ensure consistency, we consulted three industry professionals to independently re-annotate 20% of the dataset. We observed a flag-level agreement of **81.67%**; remaining discrepancies were resolved via discussion. Rather than enforce rigid inter-rater metrics, we prioritized semantic alignment and annotator intent, given the interpretive nature of this task.

2.6 Summary

Our final dataset comprises 300 annotated PR–issue pairs from six diverse GitHub repositories, enriched with code, text, and social context. Each PR is labeled with weighted green and red quality flags to support both outcome prediction and quality reasoning using LLMs. Figure 1 provides an overview of our end-to-end pipeline, spanning from the dataset creation to LLM prompting. It encapsulates the full process, the rest of which we cover in the next sections.

3 Experiment

3.1 Model Selection

Our dataset, combining textual and code-related PR features, suits CodeLlama 7B Instruct well due to its strong handling of both code and natural language. Fine-tuned for code tasks, CodeLlama excels at reasoning over code diffs and descriptions, enabling context-sensitive PR outcome predictions (Rozière et al., 2023). Its balance of performance and computational efficiency makes it an ideal choice given our resource constraints.

3.2 Experimental Setup

As a first step, we preprocess the data by cleaning text and code fields, consolidating annotations, and ensuring all inputs are consistently formatted. We then evaluate CodeLlama 7B Instruct on our dataset using zero-shot and one-shot prompting. Each experiment provides the model with full context of the target PR—including textual and code-related fields such as title, description, code diff, comments, contributor stats, and related issues. In one-shot, a labeled example PR is included to guide prediction. The model predicts both PR outcomes and associated red and green flags across the annotated dataset.

3.3 Evaluation Metrics

To evaluate model performance on pull request outcome prediction, we compare predicted outcomes (merged or not) to ground-truth annotations using standard classification metrics, such as accuracy, precision, recall, and F1-score.

We assess performance using multiple metrics for the multi-label task of detecting red and green flags in pull requests. Per-flag precision, recall, and F1-score measure how well each flag is identified, balancing false positives and negatives. Macro F1 treats all flags equally to show balanced performance, while micro F1 highlights overall effectiveness by weighting frequent flags more. Hamming loss indicates the average error rate per PR. Flattened accuracy shows the proportion of correctly classified individual flags, and exact-match accuracy reflects the percentage of PRs with a perfectly

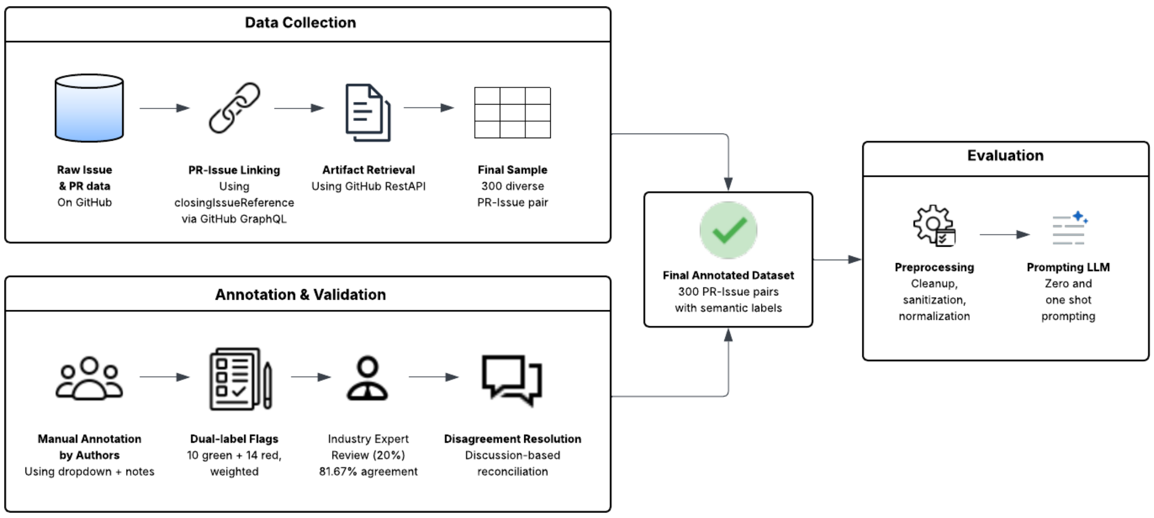
Data Collection
Raw Issue & PR data On GitHub
PR-Issue Linking Using closingIssueReference via GitHub GraphQL
Artifact Retrieval Using GitHub RestAPI
Final Sample 300 diverse PR-Issue pair
Annotation & Validation
Manual Annotation by Authors Using dropdown + notes
Dual-label Flags 10 green + 14 red, weighted
Industry Expert Review (20%) 81.67% agreement
Disagreement Resolution Discussion-based reconciliation
Final Annotated Dataset 300 PR-Issue pairs with semantic labels
Evaluation
Preprocessing Cleanup, sanitization, normalization
Prompting LLM Zero and one shot prompting

Figure 1: End-to-end pipeline from PR–issue pairing to artifact retrieval, annotation, and expert validation, followed by pre-processing and LLM prompting.

predicted flag set.

All metrics are computed using the proposed annotated dataset, with human-provided red and green flag labels as ground truth.

4 Result

We evaluated CodeLlama 7B Instruct on our dataset using zero-shot and one-shot prompting with merged and unmerged PR examples. Table 4 summarizes the performance on subsets of PRs processed after excluding cases with resource constraints.

For acceptance prediction, CodeLlama 7B Instruct achieves an accuracy of 52% in the zero-shot setting, which increases to 68% and 67% for one-shot prompting with merged and unmerged example PRs, respectively. Using a merged PR example in the prompt attains higher recall for accepted PRs, while an unmerged example provides better recall balance between accepted and unaccepted cases across the dataset. Flag prediction remains challenging, with macro F1 of 0.38 and micro F1 of 0.48 in the zero-shot setting, reflecting high recall but limited precision and overall accuracy. In the one-shot scenario, both macro and micro F1 scores drop substantially (0.01), indicating persistent difficulty in precisely identifying quality indicators. Flattened accuracy improves from 35% (zero-shot) to 69% (one-shot), but exact-match accuracy remains zero, underscoring strict evaluation criteria.

These results confirm that our dataset effectively supports PR acceptance prediction while revealing persistent challenges in multi-label flag classification.

Metric	Zero	1S-M	1S-U
Acceptance			
Accuracy	0.52	0.68	0.67
Precision (yes)	0.77	0.69	0.74
Recall (yes)	0.44	0.95	0.80
Precision (no)	0.37	0.56	0.49
Recall (no)	0.71	0.12	0.41
Macro Avg F1	0.52	0.50	0.61
Multi-label Flags			
Macro F1	0.383	0.012	0.011
Micro F1	0.481	0.008	0.011
Hamming Loss	0.65	0.31	0.31
Flattened Acc.	0.348	0.690	0.686
Exact-Match Acc.	0.000	0.000	0.000

Table 4: CodeLlama 7B Instruct performance on PR outcome and flag prediction, for zero-shot and one-shot (merged/unmerged example) prompting.

5 Conclusion

We present a novel, richly annotated dataset of 300 GitHub pull requests from six popular repositories, supporting both binary PR acceptance and multi-label quality flag prediction. Experiments with CodeLlama 7B Instruct show effective PR outcome classification but highlight challenges in fine-grained flag identification. This resource aims to facilitate further research in automated pull request review, with future work focusing on improved prompting and model fine-tuning. The dataset and details are available at this anonymous link.

Limitations

While our dataset provides a valuable resource for evaluating pull request outcomes and multi-label quality prediction, several limitations should be acknowledged. Of the 300 pull requests, predictions were successfully generated for 291 in the zero-shot setting, 263 in one-shot (merged), and 275 in one-shot (unmerged) and the remainder could not be processed due to resource constraints such as CUDA out-of-memory errors. Consequently, all reported results are based solely on these processed subsets rather than the entire dataset. Additionally, the dataset's moderate size is a consequence of the time-intensive manual annotation process, which limited the number of pull requests that could be thoroughly labeled. Finally, our evaluation employed only a single large language model (CodeLlama 7B Instruct), so the results may not generalize across other model architectures or training approaches. Addressing these limitations in future work will enhance the robustness and applicability of automated pull request review systems.

Acknowledgments

We acknowledge the use of generative AI tools, including ChatGPT and Perplexity, to assist in drafting and structuring parts of this manuscript. All final content and decisions remain the sole responsibility of the authors.

References

Muhammad Ilyas Azeem, Qiang Peng, and Qing Wang. 2020. Pull request prioritization algorithm based on acceptance and response probability. In *2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS)*, pages 231–242.

Peerachai Banyongrakkul. 2022. Multi-output learning for predicting evaluation and reopening of github pull requests on open-source projects.

Peerachai Banyongrakkul and Suronapee Phoomvuthisarn. 2023. Deeppull: Deep learning-based approach for predicting reopening, decision, and lifetime of pull requests on github open-source projects. In *International Conference on Software Technologies*, pages 100–123. Springer.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Kexing Chen, Lingfeng Bao, Xing Hu, Xin Xia, and Xiaohu Yang. 2025. E-predictor: an approach for early prediction of pull request acceptance. *Science China Information Sciences*, 68(5):152104.

Sen Fang, Tao Zhang, You-Shuai Tan, Zhou Xu, Zhi-Xin Yuan, and Ling-Ze Meng. 2022. Prhan: Automated pull request description generation based on hybrid attention network. *Journal of Systems and Software*, 185:111160.

Denae Ford, Mahnaz Behroozi, Alexander Serebrenik, and Chris Parnin. 2019. Beyond the code itself: How programmers really look at pull requests. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS)*, pages 51–60.

Georgios Gousios, Martin Pinzger, and Arie van Deursen. 2014. An exploratory study of the pull-based software development model. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, page 345–355, New York, NY, USA. Association for Computing Machinery.

Anshul Gupta. 2018. Intelligent code reviews using deep learning.

Ali Tunahan Işık, Hatice Kübra Çağlar, and Eray Tüzün. 2025. Enhancing pull request reviews: Leveraging large language models to detect inconsistencies between issues and pull requests. In *2025 IEEE/ACM Second International Conference on AI Foundation Models and Software Engineering (Forge)*, pages 168–178.

Valentina Lenarduzzi, Vili Nikkola, Nyyti Saarimäki, and Davide Taibi. 2021. Does code quality affect pull request acceptance? an empirical study. *Journal of Systems and Software*, 171:110806.

Zhongxin Liu, Xin Xia, Christoph Treude, David Lo, and Shanping Li. 2019. Automatic generation of pull request descriptions. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 176–188.

Chandra Maddila, Sai Surya Upadrasta, Chetan Bansal, Nachiappan Nagappan, Georgios Gousios, and Arie van Deursen. 2023. Nudge: Accelerating overdue pull requests toward completion. *ACM Trans. Softw. Eng. Methodol.*, 32(2).

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Defossez, Jade Copet, and Gabriel Synnaeve. 2023. Code llama: Open foundation models for code.

Md Nazmus Sakib, Md Athikul Islam, and Md Mashrur Arifin. 2024. Automatic pull request description generation using llms: A t5 model approach. In *2024 2nd International Conference on Artificial Intelligence, Blockchain, and Internet of Things (AIBThings)*, pages 1–5.

Xunhui Zhang, Yue Yu, Georgios Gousios, and Ayushi Rastogi. 2023. Pull request decisions explained: An empirical overview. *IEEE Transactions on Software Engineering*, 49(2):849–871.

Guoliang Zhao, Daniel Alencar da Costa, and Ying Zou. 2019. Improving the pull requests review process using learning-to-rank algorithms. *Empirical Software Engineering*, 24:2140–2170.