A Dataset of CodeRabbit Activities in Open Source Software Projects

Tiago Nascimento
State University of Ceará
Fortaleza, Brazil
tiago.magalhaes@aluno.uece.br

Denis Sousa
State University of Ceará
Fortaleza, Brazil
denis.sousa@aluno.uece.br

Matheus Paixao
State University of Ceará
Fortaleza, Brazil
matheus.paixao@uece.br

Italo Uchoa
State University of Ceará
Fortaleza, Brazil
italo.uchoa@aluno.uece.br

Marcio Ferreira
State University of Ceará
Fortaleza, Brazil
marcio.gabriel@aluno.uece.br

Abstract

Modern Code Review (MCR) is a fundamental software engineering practice that promotes software quality, knowledge sharing, and maintainability. Recent advances in Large Language Models (LLMs) have enabled the emergence of AI-powered code review agents capable of assisting developers by analyzing pull requests (PRs), identifying issues, and providing feedback during the review process. Among these agents, CodeRabbit has gained widespread adoption in open-source software projects. Despite this growing popularity, empirical evidence on how code review agents are configured and how they operate in real-world software engineering projects remains limited. To address this gap, we present a dataset centered on CodeRabbit, comprising agent activities collected from engineered open source software repositories. Based on 481 repositories with evidence of CodeRabbit adoption, we extracted 99,454 unique PRs and 2,255 issues. Within these artifacts, we automatically identified instances of CodeRabbit activities, including review of code changes, participation in issue discussions, generation of docstrings, generation of unit tests, resolution of merge conflicts and others. The proposed dataset enables empirical studies on AI-assisted code review, including investigations of agent configuration, participation in review discussions, authored contributions, and the feedback generated during PRs reviews. The dataset and the construction pipeline are publicly available on Zenodo [19].

Keywords

Code Review, Large Language Models, Coding Agents

1 Introduction

Modern Code Review (MCR) is a fundamental practice in software engineering consisting of a lightweight and asynchronous inspection of code changes before their integration into the codebase [1, 17, 26]. Beyond identifying defects and improving software quality, MCR facilitates knowledge sharing, improves code maintainability [27], and promotes a better collective understanding of the system among developers [21]. Nevertheless, prior studies have shown that code review still faces significant challenges, including the need to provide timely feedback and manage time constraints throughout the review process [16], as well as the high cognitive effort required from reviewers [21], particularly in large-scale projects with frequent code changes. These challenges, combined with the increasing complexity of software systems and the rapid pace of development, have motivated the adoption of automated techniques to assist developers throughout the review process [25].

GitHub has become one of the most widely used platforms for source code version control and collaborative software development [4]. In addition to supporting version management, it provides mechanisms that enable developers to perform modern code reviews. In particular, code reviews on GitHub are primarily conducted through pull requests (PRs), where proposed code changes are inspected and discussed before being merged into the codebase.

Recent advances in Large Language Models (LLMs) have enabled the emergence of AI agents capable of autonomously performing complex software engineering tasks [8]. By combining language understanding, reasoning, and tool usage, these agents are increasingly being integrated into development workflows, supporting activities such as code generation [6], testing [11], bug fixing [15], and code review [3, 10]. General-purpose coding agents, such as Claude [23], GitHub Copilot [2], and Cursor [12], are primarily designed to assist developers across a broad range of software engineering tasks. Although code review is not their sole focus, these agents are capable of analyzing code changes, and providing review feedback when prompted by developers. Differently, specialized code review agents are specifically designed to integrate with collaborative development platforms and automate review activities [3, 21].

Specialized code review agents, such as CodeRabbit[1], Qodo[2] and Codacy[3], leverage LLMs to automate code review tasks and assist developers during the review process [3, 20]. Cihan et al. [3] show that the adoption of Qodo can be effective, as 73.8% of the comments were acted upon by developers. The study also reports slight perceived improvements in code quality, particularly through earlier bug detection and code smell identification, without negatively affecting knowledge sharing within the team. Among specialized code review agents, CodeRabbit has emerged as one of the most widely adopted AI-powered code review agents. According to the GitHub Marketplace[4] listing for CodeRabbit, the agent has been installed more than 271,471 times, highlighting its growing adoption and relevance in the community.

[1] https://www.coderabbit.ai/
[2] https://www.qodo.ai/
[3] https://www.codacy.com/
[4] https://github.com/marketplace/coderabbitai

Several datasets related to coding agents have been proposed. For instance, AIDev [14] comprises 932,791 PRs produced by five agents: OpenAI Codex, Devin, GitHub Copilot, Cursor, and Claude Code. These PRs span 116,211 repositories and involve 72,189 developers. Sousa et al. [24] use PRs from AIDev to identify collaborative human-agent development projects and investigate how code clone lineages introduced by humans and coding agents survive and evolve across merged pull requests. Another example is AI Config [7], which comprises 40,585 actively maintained repositories and captures 15,591 configuration artifacts, including `Context Files` such as `AGENTS.md` and `CLAUDE.md`, as well as `Skills` and `Rules`. Despite the growing adoption of specialized code review agents, to the best of our knowledge, no publicly available dataset focuses solely on activities related to these agents.

To address this gap, this paper presents a dataset centered on CodeRabbit, comprising 99,454 unique PRs involving CodeRabbit and 2,255 issues commented on by the agent. The dataset enables empirical research on AI-assisted code review in software projects. It provides a basis for analyzing agent configuration, review participation, authored code changes, and the feedback generated during PR reviews. The dataset is publicly available at Zenodo [19].

2 CodeRabbit

CodeRabbit is an AI-powered agent specialized in code review that is designed to assist developers throughout the software development workflow. Its primary goal is to automate review activities by analyzing code changes, identifying potential issues, suggesting improvements, and interacting directly with developers. In addition to reviewing PRs, CodeRabbit is also capable of participating in issue discussions by posting comments and interacting with developers to provide clarifications, suggestions, and follow-up actions. The agent can be integrated into different development ecosystems, including GitHub. The agent offers flexible customization mechanisms, allowing users to configure its behavior either through a YAML configuration file stored within the repository or through settings available on the CodeRabbit web platform.

CodeRabbit primarily operates through PRs, where it acts as an automated reviewer integrated into the development workflow. When a PR is opened or updated, the agent analyzes the proposed code changes and generates review feedback directly within the PR discussion. Its review behavior is customizable and can be configured according to the aspects that the user wants, such as potential defects, security concerns and maintainability issues. In this work, we refer to pull requests reviewed by CodeRabbit as *Reviewed PRs*.

According to its official documentation[5], CodeRabbit provides a set of Finishing Touches features that extend its role beyond simple code review. These capabilities include **Generate Docstrings**, which analyzes modified code and generates documentation for undocumented functions and methods; **AutoFix**, which automatically implements fixes for issues identified during the review process; **Generate Unit Tests**, which creates test cases for the reviewed code; **Resolve Merge Conflicts**, which examines conflicting changes across branches and attempts to automatically produce a valid merged version; **Simplify Code**, which applies targeted refactorings aimed at improving readability and maintainability; and **Custom Recipes**, which allow teams to define reusable instructions for recurring project-specific tasks, such as enforcing coding conventions, improving type annotations, or applying repository-specific transformations.

Finishing touches can be triggered either by clicking in the specific activity (generate docstrings, generate unit tests, etc) or by invoking directly typing the agent's name (*@coderabbitai*) in a comment. Once triggered, CodeRabbit applies the requested changes using one of two mechanisms. The first consists of creating a new PR containing the generated modifications, which we refer to in this work as a *Finishing Touch PR*. Alternatively, the agent may directly commit the generated changes to the PR under review; we refer to this artifact as a *Finishing Touch Commit*.

CodeRabbit can also participate in issue discussions. Through its planning capabilities, the agent can generate implementation plans directly from issues when explicitly invoked (*@coderabbitai*) or when configured to do so automatically. Furthermore, CodeRabbit can leverage information from issue trackers and related discussions to incorporate broader project context into its recommendations.

3 Dataset Construction

3.1 CodeRabbit Presence Detection

Figure 1 illustrates the overall pipeline used to construct the dataset.

To ensure the dataset reflects real-world software engineering practices, we restricted our analysis to repositories labeled as engineered in the AI Config Dataset [7]. AI Config adopts the notion of engineered software projects proposed by Munaiah et al. [18], which characterizes repositories that exhibit evidence of active software development and maintenance, such as source code, project documentation, collaboration, and ongoing evolution, while excluding personal, toy, or non-software repositories. AI Config's repositories were initially collected using the SEART GitHub Search Tool [5], and subsequently classified according to these criteria. Following this, we selected the 36,710 repositories classified as engineered by the AI Config Dataset as the initial search space.

To identify repositories with evidence of CodeRabbit adoption, we employed a set of heuristics derived and adapted from the study by Robbes et al. [22]. We first adopted a commit-based heuristic that inspects the commit history of each repository and identifies repositories containing commits whose Git author name contains the string *coderabbit*. We also adapted the branch-based heuristic proposed by the authors by extending the search to include branches whose names start with the prefix *coderabbitai/*, in addition to the original *coderabbit/* prefix. We inspected the branch names of each repository and marked a repository as a positive match whenever at least one branch began with either of these prefixes.

In addition to these heuristics, we introduced a configuration-based detection strategy that identifies repositories containing CodeRabbit configuration files (e.g., *.coderabbit.yaml*) in the repository root. Since these artifacts explicitly configure the agent, we consider their presence to be strong evidence of CodeRabbit adoption. By combining the commit-based, branch-based, and configuration-based heuristics, we identified 481 repositories with evidence of CodeRabbit adoption, which served as the basis for the subsequent extraction stage.

[5] https://docs.coderabbit.ai/

3.2 Artifacts Extraction

After selecting repositories with evidence of CodeRabbit adoption, we employed the GitHub REST API to extract the artifacts associated with the agent, specifically leveraging the GitHub Search endpoint to apply query-based filters over PRs and issues. To capture completed and stable development activities, we restricted the extraction to merged PRs and closed issues.

Reviewed PRs were collected using the Search API query *is:pr is:merged reviewed-by:coderabbitai[bot]*. For the Finishing Touches PRs we used the query *is:pr is:merged author:coderabbitai[bot]*. Finally, we extracted closed issues containing comments from CodeRabbit using the query *is:issue is:closed commenter:coderabbitai[bot]*.

For repositories containing CodeRabbit configuration files in YAML format, we used the GitHub Contents API to retrieve the corresponding files directly from the repository structure. Each retrieved configuration file was then stored as part of our dataset, preserving both its content and associated repository context. In total, we collected 345 CodeRabbit configuration files across the selected repositories.

3.3 Activities Identification

We further analyzed the code contributions produced by CodeRabbit through its *Finishing Touches* features. To this end, we retrieved all commits associated with the previously extracted PRs in which CodeRabbit appeared as the author.

The identification process was based on regular expressions applied to commit messages, which follow recurring naming patterns generated by CodeRabbit.[6] Using these patterns, we classified commits into Finishing Touch categories such as *Generate Docstrings*, *AutoFix*, *Generate Unit Tests*, *Resolve Merge Conflicts*, and *Simplify Code*. We were not able to identify the *Custom Recipes* feature, as it allows users to define arbitrary project-specific tasks whose commit messages do not follow a standardized naming pattern, making reliable identification through regular expressions infeasible. Commits that did not match any predefined pattern were assigned to the *Other* category, allowing the dataset to accommodate previously unknown, undocumented, or custom activities.

After classifying each commit, we associated the identified Finishing Touch activity with its corresponding pull request. This enrichment enables a clear distinction between *Reviewed PRs*, in which CodeRabbit only participated by providing review feedback, and *Reviewed PRs with Finishing Touches*, in which the agent not only reviewed the proposed changes but also generated code modifications through one of its Finishing Touch capabilities. Likewise, *Finishing Touches PRs* are annotated with the specific automated activity performed by the agent, such as AutoFix or Generate Docstrings. These categories represent observable evidence of CodeRabbit activity and should not be interpreted as measures of the effectiveness or impact of the generated feedback or code changes.

4 Dataset Description

Figure 2 illustrates the dataset schema. The resulting dataset is organized as a collection of relational *.parquet* files. We adopted the Parquet format due to its columnar storage, which provides

[6]Example of a Finishing Touch commit: https://github.com/srod/node-minify/commit/659c14130a277ce6e278faeb8e64fde7fd8182c5

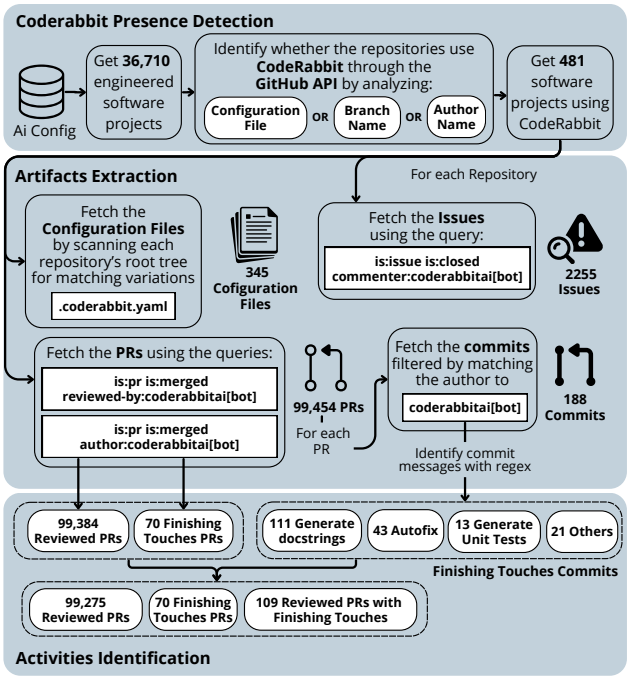

Figure 1: Overall pipeline to construct the dataset.

efficient compression and enables fast analytical queries over large-scale datasets. The dataset comprises four main tables: *repositories*, which stores repository-level metadata and CodeRabbit detection information; *pull_requests*, which contains metadata for PRs and the corresponding CodeRabbit activity labels; *issues*, which stores metadata and links for issues in which CodeRabbit participated; and *commits*, which records commits authored by CodeRabbit together with their identified Finishing Touch activity. As an additional artifact, we provide a compressed archive containing the 345 original CodeRabbit YAML configuration files extracted from repositories that expose such configurations.

Overall, the dataset contains 99,454 unique PRs collected from repositories with evidence of CodeRabbit adoption. Among them, 99,275 are *Reviewed PRs*, 70 are *Finishing Touches PRs* from 32 repositories, and 109 are *Reviewed PRs with Finishing Touches* from 56 repositories. The dataset also includes 2,255 issues containing comments from CodeRabbit and 188 commits authored by the agent. Based on the commit message classification, we identified five categories of Finishing Touch activities: 111 *Generate Docstrings*, 43 *AutoFix*, 13 *Generate Unit Tests*, 3 *Resolve Merge Conflicts*, and 18 *Other*, the latter corresponding to commits whose messages did not match any predefined activity pattern.

5 Possible Usages of Dataset

The increasing adoption of specialized code review agents has enabled new empirical studies on modern code review and human–AI collaboration [3, 20]. However, evidence remains limited on how these agents participate in review discussions and how developers respond to their feedback.

We envision the CodeRabbit dataset being leveraged in a series of studies that span not only code review, but also broader areas of

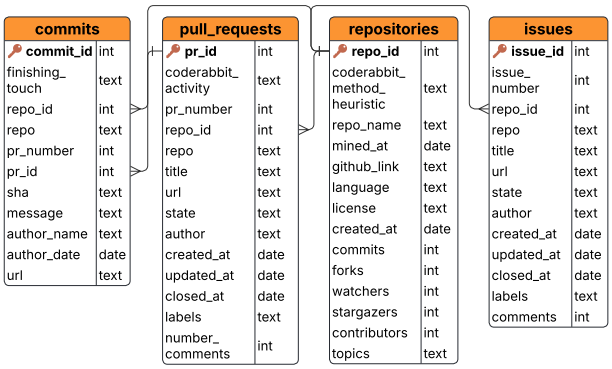

Figure 2: Dataset schema.

software engineering research, such as software maintenance, refactoring, testing, documentation, and collaborative Human-AI development. By linking CodeRabbit activities to PRs, issues, commits, and configuration files, researchers can now assess the behavior of a specialized code review agent in the context of the code changes, discussions, and contributions associated with its reviews. This enables investigation into how the agent interacts with developers, what kinds of problems it identifies, and how its recommendations and generated changes are incorporated into software projects.

During automated code review, developers and specialized code review agents produce feedback, explanations, and rationale for the changes submitted within software projects. Thus, we see CodeRabbit activity data as a scientifically valuable source of knowledge on how automated review agents identify, explain, and recommend changes during the review. This knowledge can be leveraged to answer questions that previously required direct interaction with developers, such as interviews and surveys. CodeRabbit data can be used to assess the extent to which developers address agent comments and investigate the main types of revisions it identifies, such as code smell removal, refactoring, and testing improvements. Across different systems, one may also study whether developers accept, modify, or reject changes and commits produced by CodeRabbit, providing empirical evidence of how software teams incorporate automated code review contributions into their projects.

6 Related Work

Li et al. [13] introduced *AIDev*, a large-scale dataset containing PRs exhibiting evidence of AI coding agent assistance on GitHub, along with comments, reviews, commits, and issues. The dataset enables studies on AI adoption and human-AI collaboration in software development. However, it focuses on development activities assisted by coding agents and does not investigate AI-powered code review systems such as CodeRabbit. In contrast, our dataset focuses on the activities of a code review agent throughout the review process, capturing CodeRabbit's activities across pull requests, issues, commits, and configuration files.

Galster et al. [7] presented a dataset of configuration files used by agentic AI coding tools in open-source projects, demonstrating the growing adoption of machine-readable instructions for customizing AI-assisted development workflows. In contrast, our dataset focuses on repositories adopting CodeRabbit and combines configuration artifacts with pull request review metadata, enabling future studies on AI-based code review and its potential impacts on software maintenance and quality.

Robbes et al. [22] discussed the promises and pitfalls of mining coding agent activities in software repositories. The authors showed that identifying agent contributions is inherently difficult because developers frequently integrate AI-generated suggestions before submission, motivating the need for robust heuristics and validation strategies. These findings directly motivate our methodology for identifying CodeRabbit adoption and review activities.

7 Limitations

Our dataset has some limitations that should be considered when interpreting its scope. First, although CodeRabbit supports multiple platforms, including GitHub, GitLab, and Azure DevOps, our dataset is restricted to GitHub repositories. Consequently, activities performed on other supported platforms are not represented.

Second, our repository detection strategy relies on observable traces of CodeRabbit adoption, including authored commits, branch naming patterns, and repository configuration files. However, CodeRabbit can also be configured entirely through GitHub Apps [9], without requiring repository-side configuration files or leaving identifiable artifacts such as dedicated branches or authored commits. As a result, repositories using CodeRabbit exclusively through the web interface may not be detected by our heuristics.

Finally, the identification of Finishing Touch activities is based on regular expressions applied to commit messages. While this approach successfully identifies standardized features such as *AutoFix*, it cannot reliably detect *Custom Recipes*, whose behavior and commit messages are user-defined and therefore do not follow a consistent naming convention.

8 Conclusion

In conclusion, this study presented a dataset centered on CodeRabbit, a specialized AI-powered code review agent, aimed at supporting empirical research on AI-assisted code review in real-world software engineering projects. Using engineered open-source GitHub repositories, we collected evidence of CodeRabbit adoption in 481 repositories and extracted 99,454 unique pull requests and 2,255 issues. From these artifacts, we identified and organized a diverse set of agent-related activities, including reviewed PRs, issue discussions, agent-authored commits, Finishing Touches PRs, and CodeRabbit configuration files.

The proposed dataset enables empirical studies on AI-assisted code review, including investigations of agent configuration, participation in review discussions, authored contributions, and others. As future work, we plan to extend the dataset to include more CodeRabbit artifacts and other specialized AI-powered code review agents, enabling comparative studies of their adoption and behavior in software engineering.

Artifact Availability

All artifacts, including the dataset and the construction pipeline from this study, are available in https://zenodo.org/records/21878471.

References

[1] Alberto Bacchelli and Christian Bird. 2013. Expectations, outcomes, and challenges of modern code review. In *2013 35th international conference on software engineering (ICSE)*. IEEE, 712–721.

[2] Yasharth Bajpai, Bhavya Chopra, Param Biyani, Cagri Aslan, Dustin Coleman, Sumit Gulwani, Chris Parnin, Arjun Radhakrishna, and Gustavo Soares. 2024. Let's fix this together: Conversational debugging with github copilot. In *2024 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 1–12.

[3] Umut Cihan, Vahid Haratian, Arda İçöz, Mert Kaan Gül, Ömercan Devran, Emircan Furkan Bayendur, Baykal Mehmet Uçar, and Eray Tüzün. 2025. Automated code review in practice. In *2025 IEEE/ACM 47th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 425–436.

[4] Valerio Cosentino, Javier L Cánovas Izquierdo, and Jordi Cabot. 2017. A systematic mapping study of software development with GitHub. *Ieee access* 5 (2017), 7173–7192.

[5] Ozren Dabic, Emad Aghajani, and Gabriele Bavota. 2021. Sampling Projects in GitHub for MSR Studies. In *18th IEEE/ACM International Conference on Mining Software Repositories, MSR 2021*. IEEE, 560–564.

[6] Yihong Dong, Xue Jiang, Jiaru Qian, Tian Wang, Kechi Zhang, Zhi Jin, and Ge Li. 2025. A survey on code generation with llm-based agents. *arXiv preprint arXiv:2508.00083* (2025).

[7] Matthias Galster, Seyedmoein Mohsenimofidi, Levi Böhme, Jai Lal Lulla, Muhammad Auwal Abubakar, Christoph Treude, and Sebastian Baltes. 2026. A Dataset of Agentic AI Coding Tool Configurations. *arXiv preprint arXiv:2605.08435* (2026).

[8] Matthias Galster, Seyedmoein Mohsenimofidi, Jai Lal Lulla, Muhammad Auwal Abubakar, Christoph Treude, and Sebastian Baltes. 2026. Configuring agentic AI coding tools: An exploratory study. *arXiv preprint arXiv:2602.14690* (2026).

[9] GitHub. 2026. *GitHub Apps overview*. GitHub Docs. https://docs.github.com/en/apps/overview

[10] Ahmed E Hassan, Hao Li, Dayi Lin, Bram Adams, Tse-Hsun Chen, Yutaro Kashiwa, and Dong Qiu. 2025. Agentic software engineering: Foundational pillars and a research roadmap. *arXiv preprint arXiv:2509.06216* (2025).

[11] Dong Huang, Jie M Zhang, Michael Luck, Qingwen Bu, Yuhao Qing, and Heming Cui. 2023. Agentcoder: Multi-agent-based code generation with iterative testing and optimisation. *arXiv preprint arXiv:2312.13010* (2023).

[12] Shaokang Jiang and Daye Nam. 2025. Beyond the Prompt: An Empirical Study of Cursor Rules. *arXiv preprint arXiv:2512.18925* (2025).

[13] Hao Li, Haoxiang Zhang, and Ahmed E. Hassan. 2025. The Rise of AI Teammates in Software Engineering (SE) 3.0: How Autonomous Coding Agents Are Reshaping Software Engineering. (2025). arXiv:2507.15003 [cs.SE] https://arxiv.org/abs/2507.15003

[14] Hao Li, Haoxiang Zhang, and Ahmed E Hassan. 2026. Aidev: studying ai coding agents on github. *arXiv preprint arXiv:2602.09185* (2026).

[15] Yizhou Liu, Pengfei Gao, Xinchen Wang, Jie Liu, Yexuan Shi, Zhao Zhang, and Chao Peng. 2024. Marscode agent: Ai-native automated bug fixing. *arXiv preprint arXiv:2409.00899* (2024).

[16] Laura MacLeod, Michaela Greiler, Margaret-Anne Storey, Christian Bird, and Jacek Czerwonka. 2017. Code reviewing in the trenches: Challenges and best practices. *IEEE Software* 35, 4 (2017), 34–42.

[17] Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E Hassan. 2016. An empirical study of the impact of modern code review practices on software quality. *Empirical Software Engineering* 21 (2016), 2146–2189.

[18] Nuthan Munaiah, Steven Kroh, Craig Cabrey, and Meiyappan Nagappan. 2017. Curating github for engineered software projects. *Empirical Software Engineering* 22, 6 (2017), 3219–3253.

[19] Tiago Nascimento, Denis Sousa, Matheus Paixao, Italo Uchoa, and Marcio Ferreira. 2026. *CodeRabbit Activities in Open Source Software Projects Dataset*. doi:10.5281/zenodo.21878471

[20] TMN Nimraka, MT Prabhashana, R Prainila, LL Randinu, AS Karunananda, and AMAM Adhikari. 2025. An Agentic-AI Solution for Intelligent Code Review. In *2025 9th SLAAI International Conference on Artificial Intelligence (SLAAI-ICAI)*. IEEE, 1–5.

[21] Shweta Ramesh, Joy Bose, Hamender Singh, Ak Raghavan, Sujoy Roy Chowdhury, Giriprasad Sridhara, Nishrith Saini, and Ricardo Britto. 2025. Automated Code Review Using Large Language Models at Ericsson: An Experience Report. In *2025 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 602–607.

[22] Romain Robbes, Théo Matricon, Thomas Degueule, Andre Hora, and Stefano Zacchiroli. 2026. Promises, Perils, and (Timely) Heuristics for Mining Coding Agent Activity. (2026). arXiv:2601.18345 [cs.SE] https://arxiv.org/abs/2601.18345

[23] Hélio Victor Flexa dos Santos, Vitor Costa, João Eduardo Montandon, and Marco Tulio Valente. 2026. Decoding the Configuration of AI Coding Agents: Insights from Claude Code Projects. In *Proceedings of the 2026 International Workshop on Agentic Engineering*. 63–67.

[24] Denis Sousa, Italo Uchoa, Matheus Paixao, Chaiyong Ragkhitwetsagul, and Thiago Lima. 2026. An Empirical Study of Code Clone Genealogies in Human–AI Collaborative Development. *work* 12 (2026), 20.

[25] Tao Sun, Jian Xu, Yuanpeng Li, Zhao Yan, Ge Zhang, Lintao Xie, Lu Geng, Zheng Wang, Yueyan Chen, Qin Lin, et al. 2025. Bitsai-cr: Automated code review via llm in practice. In *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering*. 274–285.

[26] Anderson Uchôa, Caio Barbosa, Daniel Coutinho, Willian Oizumi, Wesley KG Assunçao, Silvia Regina Vergilio, Juliana Alves Pereira, Anderson Oliveira, and Alessandro Garcia. 2021. Predicting design impactful changes in modern code review: A large-scale empirical study. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. IEEE, 471–482.

[27] Zezhou Yang, Cuiyun Gao, Zhaoqiang Guo, Zhenhao Li, Kui Liu, Xin Xia, and Yuming Zhou. 2026. A Roadmap for Modern Code Review: Challenges and Opportunities. *ACM Transactions on Software Engineering and Methodology* (2026).