On the Use of Agentic Coding: An Empirical Study of Pull Requests on GitHub

MIKU WATANABE, Nara Institute of Science and Technology, Japan
HAO LI, Queen's University, Canada
YUTARO KASHIWA, Nara Institute of Science and Technology, Japan
BRITTANY REID, Nara Institute of Science and Technology, Japan
HAJIMU IIDA, Nara Institute of Science and Technology, Japan
AHMED E. HASSAN, Queen's University, Canada

Large language models (LLMs) are increasingly being integrated into software development processes. The ability to generate code and submit pull requests with minimal human intervention, through the use of autonomous AI agents, is poised to become a standard practice. However, little is known about the practical usefulness of these pull requests and the extent to which their contributions are accepted in real-world projects.

In this paper, we empirically study 567 GitHub pull requests (PRs) generated using Claude Code, an agentic coding tool, across 157 diverse open-source projects. Our analysis reveals that developers tend to rely on agents for tasks such as refactoring, documentation, and testing. The results indicate that 83.8% of these agent-assisted PRs are eventually accepted and merged by project maintainers, with 54.9% of the merged PRs are integrated without further modification. The remaining 45.1% require additional changes benefit from human revisions, especially for bug fixes, documentation, and adherence to project-specific standards. These findings suggest that while agent-assisted PRs are largely acceptable, they still benefit from human oversight and refinement.

CCS Concepts: • **Software and its engineering → Integrated and visual development environments**; **Automatic programming**; *Software evolution*; *Maintaining software.*

Additional Key Words and Phrases: Agentic Coding, Coding Agent, Pull Requests, Model Context Protocol, Large Language Models

ACM Reference Format:
Miku Watanabe, Hao Li, Yutaro Kashiwa, Brittany Reid, Hajimu Iida, and Ahmed E. Hassan. 2025. On the Use of Agentic Coding: An Empirical Study of Pull Requests on GitHub. 1, 1 (September 2025), 23 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Agentic coding, defined as the use of autonomous AI agents to generate, modify, and submit code, has emerged as a transformative paradigm in software engineering. This approach is enabled by large language models (LLMs), and several agentic coding tools have recently been introduced, including Claude Code by Anthropic, Cline by the Cline team, and Codex by OpenAI. Unlike

Authors' Contact Information: Miku Watanabe, watanabe.miku.wo1@naist.ac.jp, Nara Institute of Science and Technology, Ikoma, Japan; Hao Li, Queen's University, Kingston, Canada, hao.li@queensu.ca; Yutaro Kashiwa, Nara Institute of Science and Technology, Ikoma, Japan, yutaro.kashiwa@is.naist.jp; Brittany Reid, Nara Institute of Science and Technology, Ikoma, Japan, brittany.reid@naist.ac.jp; Hajimu Iida, Nara Institute of Science and Technology, Ikoma, Japan, iida@itc.naist.jp; Ahmed E. Hassan, Queen's University, Kingston, Canada, ahmed@cs.queensu.ca.

ACM XXXX-XXXX/2025/9-ART
https://doi.org/XXXXXXX.XXXXXXX

traditional prompt-based workflows (sometimes called *vibe coding*), where developers manually guide the AI step-by-step, agentic coding enables AI agents to autonomously plan, execute, test, and iterate on development tasks with minimal human intervention.

Prior research has examined the impact of integrating LLM-based tools into software development workflows. For instance, Tufano *et al.* [38] manually analyzed 527 commits, 327 pull requests (PRs), and 647 issues, revealing that developers employ ChatGPT for a wide range of software engineering tasks, spanning 45 distinct categories, including automating the creation or enhancement of features. Similarly, Watanabe *et al.* [41] studied the use of ChatGPT in PR reviews and the corresponding developer responses. They found that 30.7% of AI-generated suggestions during code reviews were met with skepticism or disagreement, and most of them were about the generated code.

However, to the best of our knowledge, most of the previous studies focus on only LLM-based chatbots, and no prior work has investigated the impact of agentic coding on the software development process and resulting artifacts. Agentic coding tools differ from LLM-based chatbots like ChatGPT in that they can autonomously perform more complex and integrated development tasks. As these tools become increasingly embedded in collaborative development workflows, it is critical to understand their practical implications for software quality, team dynamics, and engineering processes. In this paper, we conduct an empirical study of 567 PRs created using Claude Code (hereafter referred to as Agentic-PRs) across 157 diverse open-source projects, in order to gain a deeper understanding of agentic coding. Our study focuses on these research questions (RQs):

RQ_1**: How do Agentic-PRs differ from Human PRs in terms of change size and purpose?** To better understand how developers utilize agentic tools like Claude Code, this research question investigates the distinct characteristics and underlying purposes of changes introduced by Agentic-PRs compared to Human-PRs. Our findings indicate that both Agentic-PRs and Human-PRs fix bugs and add features, but Agentic-PRs focus on non-functional improvements (tests, refactoring, documentation) while Human-PRs handle project maintenance (CI, chores).

RQ_2**: To what extent are Agentic-PRs rejected and why?** Recognizing observed dissatisfaction and the potential for extensive revisions in agent-generated code, this question explores the acceptance rate of Agentic-PRs and the specific reasons for their rejection. The study reveals that 83.8% of Agentic-PRs are accepted, though this rate is lower than Human-PRs (91.0%), with rejections primarily driven by project context, such as alternative solutions or PR size, rather than inherent AI code flaws.

RQ_3**: What proportion of Agentic-PRs are accepted without revisions? If needed, to what extent?** This question quantifies the proportion of Agentic-PRs merged without any modifications and, if revisions are needed, the extent of such changes. Results show a similar proportion of Agentic-PR (54.9%) and Human-PR (58.5%) are accepted without revisions, and when revisions are required, the effort in terms of commits, lines of code, and files modified is not statistically different between the two groups.

RQ_4**: What changes are required to revise Agentic-PRs?** Following the inquiry into the extent of revisions, this question specifically aims to identify the types of modifications most frequently necessary to refine Agentic-PRs before integration. The majority of revisions to Agentic-PRs target bug fixes (45.1%), documentation updates (27.4%), refactoring (25.7%), and code style improvements (22.1%). This implies the critical role of human oversight in ensuring the correctness, maintainability, and adherence to project standards for AI-generated code.

Replication Package: To facilitate replication and further studies, we provide the data used in our replication package.[1]

[1] https://github.com/mmikuu/OnTheUseOfAgenticCoding

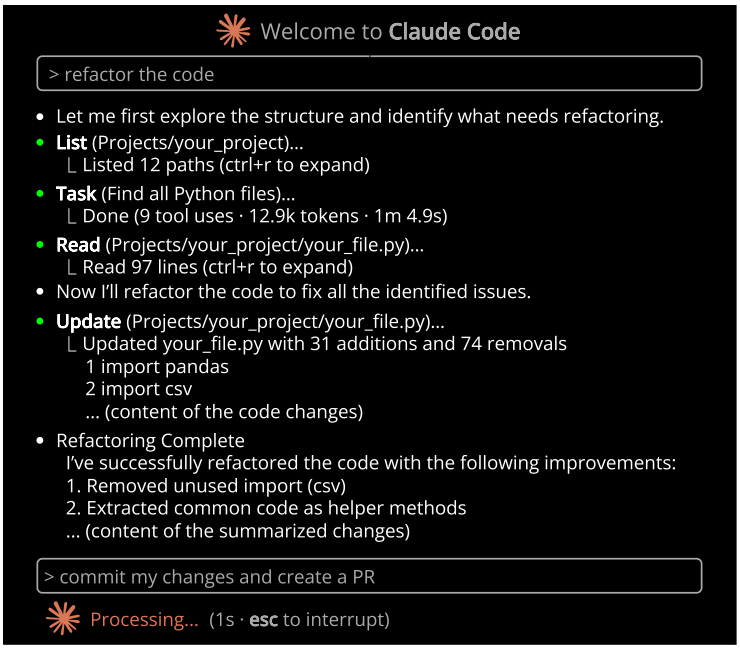

Fig. 1. Example of Claude Code refactoring and PR creation.

2 Motivating Example

Claude Code is an AI-powered assistant designed to support a broad spectrum of software engineering tasks through natural language interaction. A key innovation of Claude Code is its agentic tool use[2] and its integration with the Model Context Protocol (MCP),[3] which significantly enhances its capabilities by connecting it with diverse data sources and tools, such as filesystems, command-line interfaces, and cloud services. As of May 2025, Claude Code and agents powered by Claude 4 Sonnet have outperformed several competing approaches on SWE-bench[4] (above 70% resolution rate), a benchmark for evaluating AI performance on real-world software issues on GitHub [20].

Claude Code's core features include the ability to:[5] (i) edit files and repair bugs across the codebase; (ii) search online to find related documentation and resources; (iii) answer questions related to code architecture and logic; (iv) execute and debug tests, perform linting, and run other development commands; and (v) interact with version control systems, such as searching git history, resolving merge conflicts, and generating commits and PRs.

Fig. 1 illustrates a command-line interaction in which Claude Code performs a refactoring task. Developers can instruct Claude Code to refactor the codebase to enhance readability and reduce cyclomatic complexity. In response, Claude analyzes the files, applies appropriate transformations, such as extracting helper methods, and updates the corresponding source file. When integrated with version control systems, Claude Code can then generate a pull request (*i.e.,* an Agentic-PR) that packages the modifications for review. The PR description is automatically synthesized,

[2] https://www.anthropic.com/news/claude-3-7-sonnet
[3] https://docs.anthropic.com/en/docs/claude-code/tutorials#set-up-model-context-protocol-mcp
[4] https://www.swebench.com
[5] https://docs.anthropic.com/en/docs/claude-code
[6] https://github.com/dbieber/GoNoteGo/pull/106

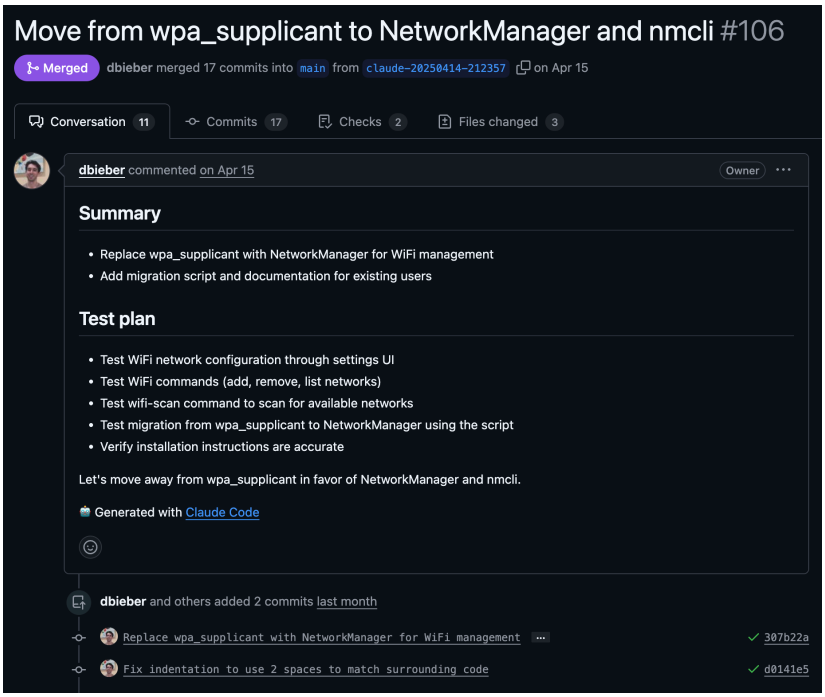

Fig. 2. Example GitHub PR created by Claude Code[6]

documenting the changes made and indicating its provenance with the phrase "Generated with Claude Code". Fig. 2 presents an example of an Agentic-PR on GitHub automatically created by Claude Code. This PR includes both the code modifications and a structured description summarizing the intent and scope of the change. Such automation lowers the effort required for developers to prepare contributions, but also raises questions about the quality and trustworthiness of these agent-generated artifacts.

Importantly, the quality of Agentic-PRs can vary. In some cases, they are immediately useful and merged with little modification. In other cases, developers express dissatisfaction. For instance, during the review process for an Agentic-PR,[7] a reviewer identified an issue in the generated code, and the author noted that the proposed change can not be used at all. In the end, the PR is closed without merging. Even when code generated by agentic coding approaches is ultimately accepted, it often requires substantial revision by developers. For example, the bottom of Fig. 2 shows how developers made several follow-up commits to improve the code originally produced by Claude Code.

These mixed outcomes motivate our study. First, we examine how Agentic-PRs differ from human ones in their content and purpose (RQ1). We then look at what happens to these PRs during review: how often they are accepted and why some are rejected (RQ2). For the accepted cases, we assess whether they are merged as-is or require additional work, and how much (RQ3). Finally, we characterize the types of changes reviewers most commonly apply when revising Agentic-PRs (RQ4).

[7] https://github.com/shopwareLabs/phpstan-shopware/pull/16

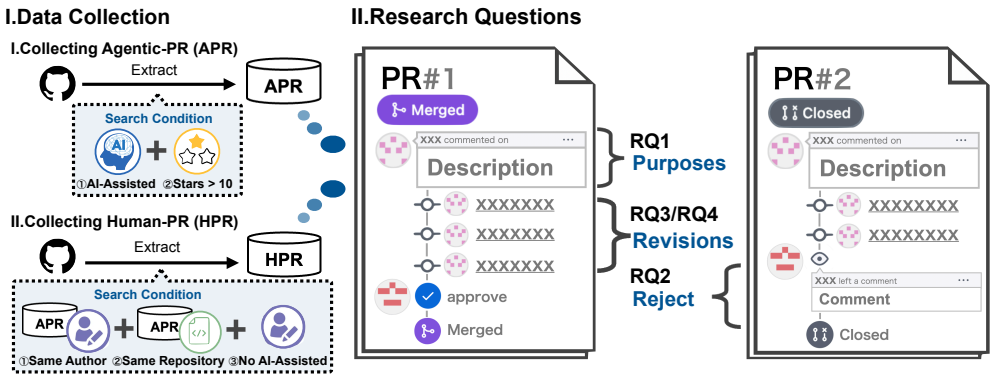

Fig. 3. Overview process of data collection

3 Data Collection

We collected Pull Requests (PRs) from open-source GitHub projects explicitly indicating that were created by Claude Code, as these represent direct artifacts of AI-assisted software development. To identify these Agentic-PRs, we searched for those containing the comment string "Generated with Claude Code" in their descriptions (as shown in Fig. 2). We conducted this search using the GitHub GraphQL API, focusing on PRs submitted between February 24, 2025 (the release date of Claude Code[8]) and April 30, 2025 (the date our study began). To ensure relevance and quality, we limited this search to repositories with at least 10 GitHub stars. This search yielded 797 PRs. We excluded 54 PRs that remained open (*i.e.,* still under development). We identified 743 Agentic-PRs after filtering.

Additionally, to characterize PRs related to agent-assisted coding, we constructed a comparison set of human-created PRs (hereafter referred to as Human-PRs). Specifically, we selected PRs created during a similar time frame and by the same author, originating from the same project repositories. To ensure a balanced comparison, we randomly sampled Human-PRs to match the number of Agentic-PRs and their authors per repository. However, due to the limited availability of matching PRs, we extended the sampling window to one year for Human-PRs. We excluded 176 Agentic-PRs from the agent-assisted coding dataset because we could not obtain the same number of Human-PRs created during a similar time frame and by the same author. In the end, both datasets have 567 PRs across 157 repositories. Importantly, we ensured that the Agentic-PR and Human-PR datasets were mutually exclusive, with no overlapping PRs.

This study defines PRs created with the assistance of agentic coding tools as *Agentic-PRs* (APRs), and those created without such assistance as *Human-PRs* (HPRs). For ease of reference, we use the abbreviations APRs and HPRs in the subsequent sections.

4 Results

In this section, we attempt to answer our research questions using the previously described dataset of PRs, detailing our approach and results for each research question.

[8]https://docs.anthropic.com/en/release-notes/claude-code

Table 1. PR purposes based on analysis of PR content

Category	Description	% APRs	% HPRs	% Δ
fix	Code changes that fix bugs and faults within the codebase	31.0%	30.8%	0.2% ↑
feat	Code changes that introduce new features to the codebase, encompassing both internal and user-oriented features	26.8%	27.6%	0.8% ↓
refactor	Code restructuring without changing its behavior, aiming to improve maintainability	24.9%	14.9%	10% ↑
docs	Updates to documentation or comments, such as README edits, typo fixes, or API docs improvements	22.1%	14.0%	8.1% ↑
test	Additions or modifications to test files, including new test cases or updates to existing tests	18.8%	4.5%	14.3% ↑
build	Changes to build configurations (e.g., Maven, Gradle, Cargo). Change examples include updating dependencies, configuring build configurations, and adding scripts	10.8%	3.6%	7.2% ↑
style	Non-functional code changes that improve readability or consistency. This type encompasses aspects like variable naming, indentation, and addressing linting or code analysis warnings	7.5%	1.8%	5.7% ↑
ci	Changes to CI/CD workflows and configurations, e.g., ".travis.yml" and ".github/workflows"	6.1%	7.2%	1.1% ↓
perf	Code changes that improve performance, such as enhancing execution speed or reducing memory consumption	5.2%	1.4%	3.8% ↑
chore	Project-wide housekeeping tasks such as dependency bumps, version increments, and miscellaneous cleanup	3.8%	10.4%	6.6% ↓

4.1 RQ_1: How do Agentic-PRs differ from Human PRs in terms of change size and purpose?

Approach. To investigate the purpose behind Agentic-PRs (APRs) and Human-PRs (HPRs), we sampled from a population of 475 merged APRs and 516 merged HPRs, and performed a manual classification on 213 APRs and 221 HPRs to understand the purpose of these PRs. We calculated the necessary sample size for each of these respective populations to satisfy a 95% confidence level with a 5% margin of error. Each PR was categorized using the two-dimensional framework proposed by Zeng *et al.* [45], which distinguishes between purpose types (fix, feat, refactor, style, perf) and object types (docs, test, ci, build, chore). Although this framework was originally developed for commit-level classification, we apply it to PR-level analysis as PR descriptions typically summarize the collective purpose of commits. Therefore, we did not apply their hierarchical rule where purpose takes precedence over object when overlaps occur, and multiple labels can be applied to each PR. The initial classification, conducted independently by the first and third authors, achieved a label-level agreement of 75.8%. Specifically, disagreements occurred on 172 out of 675 labels for Agentic-PRs and 113 out of 501 labels for Human-PRs. Since our classification was a multi-label task, we report this agreement rate instead of Cohen's Kappa [23]. To resolve disagreements, the second author reviewed all conflicting cases and proposed final labels for each disputed case. The three authors then discussed until a unanimous consensus was reached on all annotations. The first three authors have 7, 15, and 13 years of programming experience, respectively, ensuring sufficient domain expertise for accurate classification.

In addition, we examine initial PR characteristics, measuring the number of modified files, lines added and deleted, and the length of PR descriptions. Mann-Whitney U-tests ($\alpha = 0.05$) assessed whether differences between APRs and HPRs were statistically significant.

Results. ***Bug fixing is the most common category for both Agentic-PRs (***31.0***%) and Human-PRs (***30.8***%). Feature development is the second most common for both, accounting for*** 26.8***% of Agentic-PRs and*** 27.6***% of Human-PRs.*** As shown in Table 1, 66 Agentic-PRs and 68 Human-PRs were dedicated to bug fixing, making it the largest share for both. For example,

an Agentic-PR[9] corrected previously ignored HTTP request handling errors and improved linter coverage, demonstrating how AI can accelerate debugging. In addition, Agentic-PRs are also frequently used for developing features (26.8%), including some domain-specific tasks. For example, an Agentic-PR implemented functionality to store both original audio files and their corresponding WAV transcription versions, enhancing playback and transcription quality.[10]

Code refactoring is more frequent in Agentic-PRs than Human-PRs, accounting for** 24.9% **of Agentic-PRs compared to** 14.9% **of Human-PRs. Table 1 shows that 24.9% (53 out of 213) of Agentic-PRs focus on restructuring code to improve its organization and maintainability, compared to just 14.9% (33 out of 221) of Human-PRs. Developers appear to leverage agentic coding to automate boilerplate restructuring tasks. For example, PR #1964 refactored the code to switch from XML parsing using `serde` to another parser using `xml-rs`.[11] This structural modification improved parsing control and error handling mechanisms without altering the system's external behavior, as evidenced by all tests passing with unchanged output.

Test-related improvements are substantially more common in Agentic-PRs than in Human-PRs (**18.8% **versus** 4.5%**). Table 1 shows that contributions to testing are substantially more frequent in Agentic-PRs, accounting for 18.8% (40 out of 213) of PRs, compared to just 4.5% (10 out of 221) for Human-PRs. For instance, one PR[12] significantly improved test coverage from 70% to 94% by systematically adding tests for previously uncovered parts of the codebase. The agent introduced comprehensive test suites that covered previously untested code paths, including unit tests for core methods, validation for operator logic, and scenarios for error handling in SQL generation. This contribution demonstrates that agents can improve test suites not only by increasing the overall coverage percentage, but also by adding targeted tests for specific areas like individual methods and critical error handling scenarios.

Code refinement and optimization make up 12.7% of Agentic-PRs, compared to Just 3.2% of Human-PRs. According to Table 1,when combining style changes (16 vs. 4), and performance enhancements (11 vs. 3) are consistently more prevalent in Agentic-PRs. These changes often target repetitive, rule-based tasks, for example, a PR eliminating wildcard column selections to optimize database queries.[13] Similarly, stylistic cleanups, like resolving linter warnings and refining naming conventions,[14] exemplify areas where AI reduces human workload. This pattern suggests that developers leverage agents to automate such tedious, rule-based work, reducing human workload on tasks related to code conventions and micro-optimizations.

Documentation updates are more frequent in Agentic-PRs (**22.1%**) than in Human-PRs (**14.0%**), indicating significant AI involvement in maintaining textual artifacts. As shown in Table 1, documentation changes were observed in 47 out of 213 Agentic-PRs and 31 out of 221 Human-PRs. These changes include updates to user-facing documentation, code comments, and onboarding materials. For example, an Agentic-PR enhanced project documentation by adding practical code examples in docstrings, correcting formatting inconsistencies, and refining configuration descriptions to improve clarity for new users.[15]

Agents and humans contribute almost equally to maintenance and configuration tasks, accounting for 20.7% and 21.2% of pull requests, respectively. According to Table 1 when

[9] https://github.com/mattermost/mattermost/pull/30611
[10] https://github.com/rishikanthc/Scriberr/pull/71
[11] https://github.com/qltysh/qlty/pull/1964
[12] https://github.com/tomakado/dumbql/pull/17
[13] https://github.com/mattermost/mattermost/pull/30424
[14] https://github.com/Nayshins/ummon/pull/32
[15] https://github.com/brentyi/tyro/pull/266

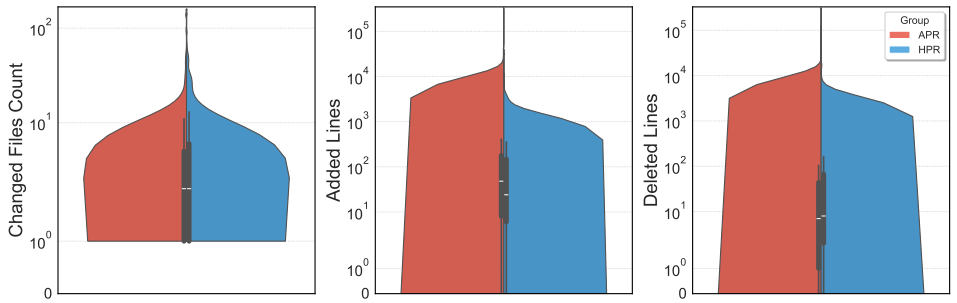

Fig. 4. Distribution of change metrics including changed files, added lines, and deleted lines in revised commits. Note that these do not include metrics from the first commit.

combining the *BUILD*, *CI*, and *CHORE* categories, Agentic-PRs account for 20.7% of these tasks, a figure very close to the 20.8% from Human-PRs. Agents demonstrate human-level performance on project-specific tasks, including dependency upgrades and workflow configuration. The ability of agents to manage complex, policy-driven work is well-exemplified by a PR[16] that enhanced CI workflows with stricter permissions and PR automation. Furthermore, their consistent assistance in routine build improvements, like updating compilation flags and frameworks[17] or performing dependency maintenance,[18] confirms that the role of agents in these areas is as significant as that of human developers.

Multi-purpose PRs are more common in Agentic-PRs, suggesting agents frequently target overlapping objectives within a single PR. Among 213 Agentic-PRs, 85 Agentic-PRs (40.0%) have multiple objectives, compared to just 27 multi-purpose Human-PRs (12.2%). This reflects an agentic coding pattern where AI consolidates repetitive or complementary tasks into a single PR. A detailed analysis of label combinations reveals that the most common pairings include *FEATURE DEVELOPMENT* combined with TEST (9.0%), *REFACTORING* with *TEST* (7.7%), and *BUG FIXES* with *TEST* (7.7%). These results suggest that agents have a consistent development of simultaneously creating and updating relevant test code when performing primary coding tasks (feature development, refactoring, and bug fixing). This allows for both code changes and their quality assurance to be achieved simultaneously within a single PR. One such example[19] updated an SDK to use a new, more powerful API endpoint (feature development). At the same time, the agent meticulously updated the corresponding test suite to validate the new functionality (test). This pattern of integrating development with immediate quality assurance showcases how agents can produce more robust and reliable changes within a single, cohesive contribution.

Agentic-PRs introduce more code and include longer descriptions than Human-PRs. Fig. 4 shows the distributions of these metrics. Examining initial submissions, both Agentic-PRs and Human-PRs modify a median of 2 files. APRs add more lines (48 vs 24) while maintaining similar deletion counts (7 vs 8). Moreover, APRs contain significantly longer PR descriptions with a median of 355 words, compared to 56 words in HPRs. Mann-Whitney U tests ($\alpha = 0.05$) confirm that the differences description length are statistically significant. One reason behind this could be that the agent is documenting its process, its reasoning, and the changes it made in detail. This could be helpful for reviewers.

[16] https://github.com/shandley/awesome-virome/pull/43
[17] https://github.com/giselles-ai/giselle/pull/687
[18] https://github.com/dxos/dxos/pull/8814
[19] https://github.com/github/github-mcp-server/pull/118

Answer to RQ1

Both Agentic-PRs and Human-PRs primarily address bug fixes and feature development. Agentic-PRs more frequently involve refactoring, documentation updates, and test improvements, while Human-PRs focus relatively more on project-specific maintenance tasks such as CI and dependency management. Additionally, Agentic-PRs are more likely to serve multiple purposes in a single submission, as well as include more code additions and longer PR descriptions.

4.2 RQ_2: To what extent are Agentic-PRs rejected and why?

Approach. We examine the proportion of Agentic-PRs (APRs) and Human-PRs (HPRs) that are eventually merged. In this study, we consider merged PRs as accepted and closed-but-unmerged PRs as rejected. We compute the merge rate as the ratio of merged PRs to the total number of APRs and HPRs (*i.e.,* including both merged and closed PRs). We also analyse the time to merge and conduct a log-rank test to evaluate the statistical significance of differences in merge times. In addition, we investigate the reasons for PR rejections by manually inspecting 92 rejected PRs.

To categorize the reasons for PR rejections, we adopted the classification framework proposed by Pantiuchina *et al.* [33], which identifies rejection patterns in refactoring PRs through empirical analysis of code review comments. Although their work focused on refactoring, the framework applies to our broader context, as many rejection reasons are independent of the change type. The framework systematically organizes rejection reasons into process-related and refactoring-specific categories. We use five subcategories: (i) three from process-related subcategories called *Are inactive (author/community)*, *Are too large*, and *Are obsolete* ; and (ii) two from refactoring-specific subcategories called *Do not add value* and *Contain choices of non-optimal design solutions*. The classification process by two independent authors achieved 68.5% agreement between the two primary annotators, with a Cohen's K coefficient of 0.4865, indicating moderate agreement [23].

The selection of these categories was motivated by two factors: First, preliminary analysis of our dataset revealed that these rejection patterns occurred with high frequency. Second, employing an empirically-validated classification enhances the reliability of our findings and facilitates comparison with existing literature. However, during our analysis, we identified rejection reasons that could not be adequately classified within the framework from Pantiuchina *et al.* [33]. For these cases, we inductively derived new categories through open coding of rejection comments, thereby extending the original classification to comprehensively capture the rejection patterns specific to our context.

Results. 83.8*% of Agentic-PRs are accepted by human reviewers, although their acceptance rate remains statistically lower than Human-PRs (*91.0*%).* Table 2 shows that 475 out of 567 APRs were merged, compared to 516 out of 567 HPRs. A chi-squared test revealed that Human-PRs have a statistically significantly ($p < 0.05$) higher acceptance rate than Agentic-PRs. However, a

Table 2. Acceptance and merge time statistics of PRs.

	# Accepted	# Rejected	# Total	% Acceptance	Median time to merge
APRs	475	92	567	83.8%	1.23 hours
HPRs	516	51	567	91.0%	1.04 hours

log-rank test shows no significant difference in merge times between Agentic-PRs and Human-PRs, suggesting comparable review efficiency for AI-generated contributions. The median time to merge for APRs is 1.23 hours, while HPRs take 1.04 hours.

The most common reasons for Agentic-PR rejections stem from project evolution and PR complexity, not just code quality. Table 3 shows that *ALTERNATIVE SOLUTIONS* (12.1%), *OBSOLESCENCE* (3.3%), and *OVERSIZED PRs* (3.3%) frequently result in Agentic-PR rejections, reflecting project evolution and maintainability concerns. For example, one Agentic-PR[20] was closed after the team resolved the issue using a different solution, stating, "*We might come back to this, but for now solving the underlying question in a different way.*" Another PR[21] became obsolete after a newer contribution addressed the same functionality. Similarly, a large PR[22] was closed with the comment, "*Closing in favor of smaller, more focused PRs to make reviews more manageable*," emphasizing the difficulty of integrating oversized contributions into collaborative review processes.

Process-related issues such as verification-only submissions (5.5%) and merge conflicts (1.1%) also contribute to rejections in Agentic-PRs. Table 3 shows that 5 rejected APRs were created solely for automated checks, 1 encountered costly merge conflicts. For example, one verification-only PR[23] was closed after confirming intended CI processes. A different PR[24] faced extensive merge conflicts the contributor could not resolve, prompting abandonment. These cases illustrate how agentic coding contributions can be rejected for logistical or procedural reasons common in development workflows, making this category the second most frequent cause.

Technical shortcomings, such as issues with code implementation quality rather than strategic choices, account for 4.4% of Agentic-PR rejections, with 2.2% rejected for non-optimal design solutions, 1.1% for overly complex implementations, and 1.1% for introducing bugs or breaking functionality. Table 3 shows that 4.4% of the rejected Agentic-PRs are due to technical reasons where the implementation approach or code quality was problematic. For example, one PR[25] was closed for bypassing project-specific serialization mechanisms, violating architectural principles. Another PR[26] was rejected after reviewers found that existing themes could achieve the same functionality with reduced complexity. Furthermore, a PR[27] closed with the comment "*this is badly extracted and has some hallucinations, new CSS changes, and more importantly breaks the UI*," demonstrating functional failures. These examples highlight persistent limitations in AI's ability to generate quality code that maintains system integrity.

Strategic misalignment, such as issues with what to build rather than how to build it, accounts for 2.2% of Agentic-PR rejections, with 1.1% closed for not adding value and 1.1% for not aligning with community interests. Table 3 indicates these PRs targeted wrong problems or proposed unnecessary changes, regardless of implementation quality. For instance, one PR[28] was closed after maintainers determined the changes, while technically correct, failed to address the actual performance bottlenecks that mattered. Another PR[29] was rejected after reviewers referenced the project's policy document, determining the proposed large-scale changes

[20] https://github.com/mattermost/mattermost/pull/30593
[21] https://github.com/OpenAdaptAI/OpenAdapt/pull/946
[22] https://github.com/solvespace/solvespace/pull/1553
[23] https://github.com/giselles-ai/giselle/pull/599
[24] https://github.com/openai/codex/pull/612
[25] https://github.com/zenml-io/zenml/pull/3375
[26] https://github.com/pytorch/test-infra/pull/6410
[27] https://github.com/pytorch/test-infra/pull/6400
[28] https://github.com/mattermost/mattermost/pull/30890
[29] https://github.com/MaiM-with-u/MaiBot/pull/195

Table 3. Reasons for Agent PR rejection

Category	Description	% APRs
Are implementated by other PRs/developers	The contributor or project team chooses a different solution before this PR can be merged	12.1%
Submission for verification	The PR is created solely to trigger automated checks (e.g., CI pipelines) and is not intended for merging	5.5%
Are too large	The PR is too large or complex, making effective review impractical	3.3%
Are obsolete	The proposed changes become outdated or irrelevant due to evolving project requirements or newer implementations	3.3%
Are inactive (author/community)	A project's state of ceased or minimal development activity, often implying a lack of ongoing maintenance or community engagement	2.2%
Contain choices of non-optimal design solutions	The PR implements a design that is considered suboptimal, inefficient, or architecturally unsound	2.2%
Do not add value	The PR provides no clear or significant benefit to the project, its users, or its maintainers	1.1%
Increase complexity	The proposed solution was more complex than warranted	1.1%
No confidence in AI-generated code	The PR is rejected because the code was AI-generated and lacks sufficient human review or understanding to ensure reliability	1.1%
Introduce bugs/ break APIs / breaks compatibility	The PR introduces bugs into existing functionality, breaks API compatibility, or includes changes that disrupt normal system operation	1.1%
Are not in the community interest	The PR does not align with the project's direction or community goals, and is judged not worth the investment of limited resources	1.1%
Introduce merge conflicts	Resolving the merge conflicts requires significant manual effort, beyond a simple rebase	1.1%
Not sure	The PR has review comments, but they are ambiguous or open to multiple interpretations, making it difficult to classify the specific rejection reason	1.1%
Unknown (No feedback provided)	The PR was closed without explanatory comments or discussion, preventing classification of the actual rejection reason	63.7%

did not provide sufficient benefit to justify their inclusion. These cases demonstrate challenges in AI's understanding of project priorities and identifying which problems actually need solving.

Contributor inaction accounts for** 2.2**% of Agentic-PR rejections, with PRs abandoned when contributors stop responding to the review process. Table 3 indicates that 2 rejected APRs were closed due to general inactivity. For instance, a PR[30] was marked stale and automatically closed after the contributor failed to respond to implementation concerns. Another case[31] involved a reviewer commenting "Waiting on this to be resolved" with a link to specific code review feedback, but the contributor never addressed the issue, resulting in automated closure by a bot. These examples illustrate how Agentic-PRs, when not actively managed by human contributors, face higher rejection risks due to minimal post-submission engagement.

Trust remains a barrier for Agentic-PRs, with** 1.1**% of rejected APRs explicitly closed due to lack of confidence in their correctness. As shown in Table 3, 1 rejected APRs was withdrawn because contributors or maintainers doubted the reliability of AI-generated code. For instance, one contributor closed their PR[32], stating explicit concerns about validating Claude's output. This reflects a key socio-technical barrier, where agent-assisted development outpaces human trust in AI capabilities, limiting the integration of such contributions despite technical potential.

[30] https://github.com/osmosis-labs/osmosis/pull/9029
[31] https://github.com/ApeWorX/ape/pull/2532
[32] https://github.com/beekeeper-studio/beekeeper-studio/pull/2962

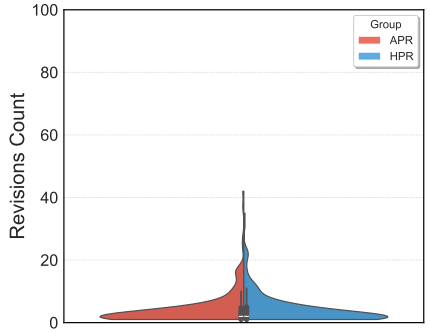

Fig. 5. The number of revisions commits in Agentic-PRs and Human-PRs. The inner box shows the interquartile range, and the white lines indicate the medians.

The majority (63.7%) of rejected Agentic-PRs were closed without explanatory comments or discussion. Table 3 shows that 58 out of 91 rejected APRs fell into this category. For instance, a PR[33] was closed without discussion or review comments, leaving the rejection reasons entirely unknown. This lack of feedback highlights a transparency challenge when evaluating why AI-generated contributions are declined.

Answer to RQ2

Agentic-PRs are widely accepted in real-world projects, with a 83.8% acceptance rate and similar review times to Human-PRs. However, their acceptance is lower than human-written PRs (91.0%). Most rejections stem from project context, such as the changes being implemented by other developers, submissions made purely for verification, or the PR's size, rather than inherent flaws in AI-generated code. Still, technical issues (e.g., suboptimal design, complexity) and trust concerns persist.

4.3 RQ_3: What proportion of Agentic-PRs are accepted without revisions? If needed, to what extent?

Approach. To understand the extent to which developers revise Agentic-PRs, we conducted a two-step analysis. First, to identify the proportion of agent-generated changes accepted without any modification, we calculated the percentage of Agentic-PRs and Human-PRs that were merged with a single commit, containing only the initial submission. This metric allows us to quantify the rate at which agent-assisted changes are deemed sufficient without further developer intervention. Second, for the subset of APRs that underwent at least one revision, we analysed the subsequent revision effort required to get the PR merged. We quantified this effort using the following four metrics, measured from the second commit to the final commit in the PR: (1) the number of revision commits, (2) the number of files changed, (3) the total number of lines added/deleted, (4) the percentage of code modification compared to the initial submission. This analysis provides insight into the cost and nature of developers' modifications to agent-generated code.

Results. ***The majority of both Agentic-PRs and Human-PRs are merged without revision.*** Among merged PRs, we observe that 54.9% of APRs (261 PRs) are merged as-is with a single commit,

[33] https://github.com/anthropics/anthropic-cookbook/pull/145

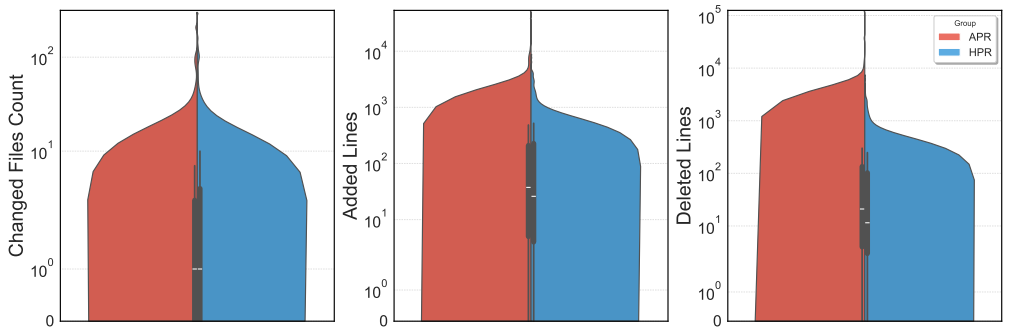

Fig. 6. Distribution of change metrics including changed files, added lines, and deleted lines in revised commits. Note that these do not include metrics from the first commit.

as compared to 58.5% (302 PRs) of HPRs. In addition, the chi-squared test revealed no statistically significant difference between the two groups ($p > 0.05$). This similarity indicates that, in many cases, changes proposed by agentic coding are accepted without further human intervention, suggesting a baseline level of trust and adequacy in the initial agent-generated code.

When revisions are required, there is no statistically significant difference in revision frequency or cost. Analysis of revision patterns shows that both APRs and HPRs typically include a median of 2 revised commits prior to merge (see Fig. 5). The Mann-Whitney U tests show no statistically significant differences between Agentic-PRs and Human-PRs in terms of revision frequency. Although the median number of files, lines added, and lines deleted in revised commits can differ between the two groups (see Fig. 6), none of these differences are statistically significant at $\alpha = 0.05$. These results suggest that the revision cost for Agentic-PR is not significantly different from that for Human-PR. Given this equivalence, leveraging AI to automate the initial stage of PR creation could effectively reduce developer burden and improve productivity.

When comparing revisions against the baseline (*i.e.,* the first submission), the additionally changed files represented a median increase of 50.0% for both APRs and HPRs. For changed lines, HPRs showed larger increases with a median of 121.1% compared to 94.3% for APRs.[34] This suggests that the revision patterns between AI-generated and human-written PRs are statistically similar.

Answer to RQ3

The proportion of Agentic-PRs and Human-PRs that did not require revisions was similar, at 54.9% and 58.5%, respectively. Furthermore, a Mann-Whitney U test on the revision effort measured by the number of commits, lines of code added/deleted, and files modified revealed no statistically significant difference between Agentic-PRs and Human-PRs.

4.4 RQ_4: What changes are required to revise Agentic-PRs?

Approach. We examined all 475 merged Agentic-PRs and identified 214 Agentic-PRs (45.1%) that had modifications between initial submission and merge. We performed manual classification on these 214 Agentic-PRs to understand what types of changes were made during the review process.

[34]Some pull requests contain zero added or deleted lines in their initial submission, making it impossible to compute relative metrics based on additions or deletions alone. Therefore, we use the total number of modified lines, defined as the sum of added and deleted lines.

Table 4. Revision based on analysis of PR content

Category	Description	% APRs
fix	Code changes that fix bugs and faults within the codebase	45.1%
docs	Updates to documentation or comments, such as README edits, typo fixes, or API docs improvements	27.4%
refactor	Code restructuring without changing its behavior, aiming to improve maintainability	25.7%
style	Non-functional code changes that improve readability or consistency. This type encompasses aspects like variable naming, indentation, and addressing linting or code analysis warnings	22.1%
chore	Project-wide housekeeping tasks such as dependency bumps, version increments, and miscellaneous cleanup	19.9%
test	Additions or modifications to test files, including new test cases or updates to existing tests	15.5%
feat	Code changes that introduce new features to the codebase, encompassing both internal and user-oriented features	14.6%
build	Changes to build configurations (e.g., Maven, Gradle, Cargo). Change examples include updating dependencies, configuring build configurations, and adding scripts	13.3%
ci	Changes to CI/CD workflows and configurations, e.g., ".travis.yml" and ".github/workflows"	6.6%
perf	Code changes that improve performance, such as enhancing execution speed or reducing memory consumption	1.3%

We manually inspect all commits added after the initial submission and classify them using the same two-dimensional framework from Zeng *et al.* [45] described in Section 4.1. For each PR, we aggregated revision types at the PR level to avoid over-representing PRs with numerous small commits. For example, if a PR included three bug fixes and two stylistic changes across separate commits, it was counted as one instance of each change type. This manual classification was performed by the second and third authors. As in Section 4.1, we treated this as a multi-label task and measured agreement at the label level. The overall agreement rate was 64.9% (541 out of 844 labels). Disagreements were resolved by the second author following the same process described in Section 4.1. In addition, to understand the extent of continued AI assistance during the revision process, we analyzed co-authorship patterns in revision commits. We extracted commit metadata to identify commits containing "`Co-Authored-By: Claude`" attribution.

Results. ***The majority (45.1%) of revisions to Agentic-PRs target bug fixes.*** As shown in Table 4, the most common revision type was fixing functional bugs, accounting for 45.1% (102 out of 226) of revised Agentic-PRs. These fixes typically involved critical error handling improvements, and representative revisions addressed situations where critical errors were improperly handled. In one case,[35] concurrent error propagation required introducing channel-based communication mechanisms to ensure fatal errors were immediately surfaced from worker closures. Similarly, another revision[36] upgraded file operation failures from warnings to fatal errors, recognizing that certain failure conditions should halt execution rather than proceed with a potentially corrupted state. These patterns reveal that agent-generated code frequently implements optimistic error handling strategies that fail to distinguish between recoverable and non-recoverable failure conditions.

Addressing documentation gaps constitutes the second largest focus (27.4%) of Agentic-PRs revisions. Documentation updates were present in 27.4% (62 out of 226) of revisions, as shown in Table 4. Although agents sometimes generated or updated code comments, they often failed to synchronize all relevant artifacts. For example, one revision[37] removed an outdated optional dependencies section from installation documentation that should have been deleted alongside related code changes. This disconnect means that, in practice, human reviewers spend a non-trivial

[35] https://github.com/qltysh/qlty/pull/1591/commits/87444f54e702a64f519ca1052380bf9f50f7be96
[36] https://github.com/qltysh/qlty/pull/1588/commits/a38e75cc15657d8864cff66b388189b1ea677e19
[37] https://github.com/brentyi/tyro/pull/266/commits/8d3bf92a48a8bef4dbb09bfc31bc39dbfb544046

amount of effort ensuring that documentation, README files, and code comments accurately reflect code changes from agents.

Refactoring is frequently required (*25.7*%) to improve code structure before merging Agentic-PRs. Table 4 shows that 58 out of 226 modified Agentic-PRs involved refactoring, highlighting the need for structural improvements in these submissions. These structural changes, which do not affect program behaviour, were frequently necessary to eliminate code duplication, enhance modularity, or revise architectural decisions made by the agent. For example, a reviewer consolidated redundant initialization logic spread across multiple entry points into a unified service.[38] In this case, common functionality for database migrations and file synchronization was extracted from separate components and moved into a shared module, improving both error handling and logging consistency. This finding suggests that although the initial submissions from Agentic-PRs were functionally correct, reviewers often had to refactor them to align better with project architecture, enhance maintainability, and reduce technical debt.

A common portion (*22.1*%) of revisions is commonly focused on polishing code style rather than behaviour. As shown in Table 4, 50 out of 226 modified Agentic-PRs received style-related revisions during review, indicating code quality issues in initial submissions. Typical fixes include enforcing naming conventions, correcting formatting, and resolving linter warnings that agentic tools missed. For example, a reviewer addressed multiple static analysis violations that were present in the agent's original code,[39] including unused imports, exception variables that were declared but never referenced, and incorrect import ordering. These changes are largely cosmetic but necessary for integration, and suggest that current agents often underperform on project-specific style rules, requiring human intervention to maintain readability and consistency. Static analysis violations and ignored best practices were common triggers for these types of revisions.

Project Housekeeping tasks account for* 19.9*% of revisions. According to Table 4, chores represent 19.9% (45 out of 226) of revisions, typically involving project-wide metadata such as version bumps or release notes that the agent overlooked. For example, one revision[40] involved a simple but critical version bump from "3.0.0-alpha01" to "3.0.0-alpha02" in a Gradle configuration file (`libs.versions.toml`), a necessary step for a new release that the agent did not perform. This pattern indicates a specific gap in agent capabilities: while they successfully modify application code, they often fail to propagate corresponding changes to project-level configuration files. Consequently, these essential administrative tasks fall to human reviewers to ensure the project's versioning and release process remains consistent and accurate.

New features are added in* 14.6*% of Agentic-PRs by human reviewers. Table 4 indicates that 33 out of 226 modified Agent PRs (14.6%) received feature additions during the review process. For example, one revision[41] expanded the PR review API functionality that the agent had initially implemented with basic capabilities. The revision added support for multi-line comments, including new parameters for line positioning, validation logic for parameter combinations, and comprehensive test coverage. This enhancement demonstrates that agent-generated implementations often provide core functionality but miss advanced features or edge cases that developers identify during review, requiring subsequent additions to meet complete user requirements.

Build configuration adjustments occur in* 13.3*% of reviewed Agent PRs. Table 4 shows that 30 out of 226 modified Agentic-PRs received build configuration updates during review. One

[38] https://github.com/basicmachines-co/basic-memory/pull/88/commits/ca5fa557c25bead9aafcf545544957d835bba798
[39] https://github.com/INCATools/ontology-access-kit/pull/831/commits/5a6f71986fcfaafa291b9aa405d51d76c8c27c97
[40] https://github.com/yumemi-inc/Tart/pull/106/commits/88594d8a116e7c91390740cf7e9ddb7ccfaa61f5
[41] https://github.com/github/github-mcp-server/pull/118/commits/0527bc552591df807ee7be0b3a79f428d2f0ed2c

revision[42] addressed compatibility issues between the linting tool and the newer Go version. The agent's initial implementation did not account for version incompatibilities, causing build failures with Go 1.24.1. The revision updated the linter version, configured it to run with a reduced set of checks, and modified the build process to continue despite linting failures. This temporary solution allowed the build pipeline to function while a comprehensive fix was planned for a subsequent update. The revision illustrates that agent-generated build configurations may overlook toolchain compatibility requirements, requiring manual intervention to maintain functional build processes across version updates.

Human reviewers also shore up test coverage that agents frequently omit. Although tests account for only 15.5% (35 out of 226) of the revisions in Table 4, the added suites are often substantial and guard edge cases and failure paths that the original submission did not address. For instance, one revision[43] added comprehensive unit tests for GPU X-VGA support detection functionality that were absent from the agent's initial submission. The new tests cover device-argument generation, detection-process validation, error-handling scenarios, and configuration-flow verification. The agent had implemented the GPU support feature without corresponding test coverage, which reviewers identified as a gap requiring remediation. This pattern echoes our earlier finding in Section 4.2 that reviewers explicitly reject Agentic-PRs when they cannot fully trust the code without further human verification.

CI/CD and performance optimizations represent a smaller but crucial tail of revisions (7.9%). CI/CD modifications are relatively rare, comprising 6.6% of revisions, yet they are essential for maintaining a healthy pipeline. One revision, for instance, introduced a script to enforce version-string consistency across build artifacts after the original agent submission broke continuous integration.[44] Performance improvements appear in an even smaller fraction, just 1.3%, such as when a reviewer added a cache-busting mechanism to ensure data freshness for an efficient but potentially stale storage driver implemented by an agent, thereby achieving both accuracy and performance.[45] Taken together, these results highlight that while agentic coding already addresses the majority of functional and stylistic changes, human reviewers continue to play a key role in ensuring project hygiene, pipeline reliability, and targeted performance optimization.

Agents remain actively involved in 41.1% of all revisions for Agentic-PRs. Our analysis shows that 41.1% (88 out of 214) of the revised Agentic-PRs were co-authored with Claude, accounting for 34.1% (298 out of 873) of the commits made during the revision process. This indicates that developers frequently rely on AI tools not only for initial code generation but also for iterative refinement during review. The substantial proportion of AI co-authorship in revisions underscores the sustained role of agentic systems throughout the software development cycle.

Answer to RQ4

Among merged Agentic PRs, 45.1% require reviewer revisions, primarily for bug fixes (45.1%), documentation updates (27.4%), refactoring (25.7%), and code style improvements (22.1%). This highlights that while AI-generated code is a strong starting point, human oversight is essential to ensure correctness, maintainability, and adherence to project standards.

[42] https://github.com/coder/coder/pull/17035/commits/3c3aa219b895a981dd13a1b530603aa257f5549e
[43] https://github.com/aleph-im/aleph-vm/pull/786/commits/bf69485b01c6d246126bd471f5f1509eef4cc84d
[44] https://github.com/coder/coder/pull/17164/commits/de29681960767cb7eb01b407fa6de4fa48a07976
[45] https://github.com/giselles-ai/giselle/pull/695/commits/f9356079fe49ce56ca04e821098432916367032b

5 Implications

In this section, we discuss the implications of our findings for researchers, developers, and coding agent builders.

5.1 Implications for researchers

Quantify the socio-technical cost of building trust in Agentic-PRs. Although only 2.2% of rejected Agentic-PRs explicitly cited "lack of confidence," many rejections followed slow reviews or repeated revision requests (Section 4.2), suggesting that reviewer skepticism toward AI-generated code operates more subtly than direct statements might indicate. This hidden friction represents a significant barrier to AI tool adoption that traditional metrics fail to capture. Mixed-method studies that combine review-time analytics with developer interviews could reveal how latent scepticism increases cognitive load, and whether transparency cues (*e.g.,* provenance statements) mitigate this effect. In addition, given that nearly one-third of merged Agentic-PRs still required bug fixes, style adjustments, or refactoring (Sections 4.3 and 4.4), measuring discussion length, reviewer effort, and escaped-defect rates will help identify when AI contributions save work versus when they add overhead.

Create PR-centric benchmarks that reward alignment across code, documentation, and review resolution. Current benchmarks such as SWE-bench [20] overlook documentation drift, style adherence, and reviewer negotiation, all of which accounted for 48.5% of Agentic-PR revisions (Section 4.4). This gap between benchmark evaluation and real-world requirements represents a fundamental misalignment in how we assess coding agent capabilities. While existing benchmarks focus predominantly on functional correctness, whether the generated code passes test cases, they fail to capture nearly half of the actual work required to integrate AI-generated code into production systems. We suggest building datasets that incorporate time-to-merge, rework size, and the alignment between documentation and implementation. Such benchmarks can be applied during both training and evaluation of coding agents to promote improvements beyond functional correctness.

Instrument the entire human-AI workflow, not isolated tools. Our results show that 41.6% of revision commits were co-authored by Claude Code (Section 4.4), and review threads frequently referenced GitHub Copilot and other agents. This widespread multi-agent collaboration highlights a critical gap in our understanding of agent-assisted development workflows. We recommend that future studies collect comprehensive full-trace datasets that capture the entire development lifecycle from initial prompts and agent planning phases to intermediate code generations, CI/CD logs, review iterations, and final merge decisions. Such datasets would reveal crucial interaction patterns, including where hand-offs between agents and developers occur, where duplicate efforts waste resources, and where conflicting edits from different agents inflate latency and introduce defects.

5.2 Implications for developers

Split large, multi-purpose PRs into smaller, easy-to-review submissions. Over 27% of Agentic-PRs combined multiple tasks (Section 4.1), and "too large" was among the top three rejection reasons (Section 4.2). Large PRs increase reviewer cognitive load and raise the risk of rejection, as previous studies [42] have warned developers for decades. This challenge remains relevant even for agents. In fact, RQ1 shows agents generate comprehensive solutions that attempt to address multiple issues simultaneously. The tendency of AI coding agents to produce extensive changes when given broad directives mirrors the problematic patterns seen in human development, but with potentially greater scale and complexity. When assigning a task to a coding agent, developers should adhere

to the principle of providing a small, self-contained task that can be addressed in a single PR, or structure broader tasks to produce a sequence of smaller, follow-up PRs for wide-reaching changes.

Embed project-specific style, rules, and architecture in agent instructions. Style mismatches accounted for 22.1% of revisions, while refactors accounted for another 25.7% (Section 4.4). Missing documentation and tests contributed 21.7% and 13.7%, respectively. These statistics reveal a critical pattern: much of the revision work on AI-generated code results not from functional errors but from integration friction, which is the gap between what agents produce and what teams expect. This misalignment creates substantial hidden costs, as developers must repeatedly guide agents toward project-specific conventions that could have been specified upfront. We recommend that developers maintain a dedicated guideline file (*e.g.,* `CLAUDE.md`, supported by Claude Code) that includes formatting rules, design principles, and architectural constraints. This practice can help coding agents produce outputs aligned with project standards, ensuring that code, documentation, and tests remain synchronized as part of the definition of done.

5.3 Implications for coding agent builders

Coding agent builders should present uncertainty and supply review scaffolding, not just code. RQ2 (Section 4.2) shows that PRs rejected for non-optimal design often lacked implementation rationale. This absence of reasoning transparency creates a critical bottleneck in the review process, as reviewers must expend significant effort inferring why certain design decisions were made, or worse, may reject sound solutions simply because the underlying logic remains opaque. We recommend attaching a "confidence card" to each PR, including the plan followed, key assumptions, considered alternatives, and known edge cases. A reviewer checklist could guide targeted human scrutiny.

Automate low-risk PR maintenance tasks such as rebases, conflict resolution, and stale-response handling. High-cost conflict resolution accounted for 1.1% of rejections (Section 4.2). These rejections often occur not because of fundamental code issues but due to timing misalignments. In other words, PRs that were valid when created become unmergeable as the target branch evolves. Currently, developers must manually monitor agent-generated PRs, performing routine maintenance tasks that consume time without adding meaningful value. Agents could be enhanced to periodically rebase branches against the main branch, resolve simple conflicts, and automatically respond to staleness warnings. These capabilities would reduce manual oversight and help keep PRs mergeable.

Coding agent builders should integrate additional tools to improve adherence to project conventions. The high rates of style (22.1%) and refactoring (25.7%) revisions indicate that generic models do not fully capture local conventions (Section 4.4). This also implies that creating a persistent friction between agent capabilities and team expectations. This misalignment stems from agents being trained on diverse codebases with conflicting conventions, making them unable to infer project-specific patterns from limited context. Using the Model Context Protocol (MCP) servers [14], builders can connect agents to linters (*e.g.,* `pylint`) and static analysis tools to enforce formatting and structural rules before submission. Similarly, build-checking and test-coverage tools could be integrated to catch common revision triggers in advance.

6 Related Work

6.1 Human-AI collaboration in software engineering

Recent research has emphasized the increasing collaboration between human developers and AI-powered technologies in common software engineering tasks [22], such as requirements documentation [35], code search [25], and code generation [7]. Hassan *et al.* [15] characterized this

collaborative relationship within the emerging paradigm of Software Engineering 3.0, envisioning AI teammates that enhance developer productivity and reduce the cognitive load on human developers. However, despite these technical advances, human oversight remains crucial. AI-powered tools have demonstrated potential for improving productivity, but human-in-the-loop approaches are still required to maintain high-quality results [6, 28]. Hence, establishing clear guidelines and best practices for managing human-AI interactions remains an essential step for industrial organizations to integrate these technologies [36].

In addition, multi-agent frameworks such as LLM-Agent, FlowGen and FightFire assign specialised roles (planner, coder, tester) to cooperating LLMs, emulating Agile practices and raising robustness [16, 24, 44]. A recent systematic review consolidates prompt-engineering techniques and taxonomies of human-AI collaboration scenarios in software engineering [17].

However, despite these technical achievements, significant social and human challenges exist in AI-assisted development. Piorkowski *et al.* [34] and Nahar *et al.* [29] uncover persistent knowledge gaps inside multi-disciplinary AI teams, reinforcing the call for explicit interaction guidelines. Furthermore, Adam *et al.* [1] found that LLM-assisted reviews can disrupt collective accountability mechanisms inherent in traditional peer reviews, as developers value intrinsic factors such as professional integrity and reputation that cannot be maintained when interacting with AI systems. These studies highlight the complex socio-technical challenges in human-AI collaboration.

6.2 AI-assisted issue reports and PRs

Previous studies have explored AI-assisted activities related to issue reports [3, 21] and PRs [5, 41]. However, no prior research has investigated the use of agentic coding tools, such as OpenAI's Codex[46] and Claude Code, in real-world development contexts. Instead, current research primarily focuses on LLM-based chatbots like ChatGPT. For example, Chouchen *et al.* [5] reported that ChatGPT is frequently used in review-intensive PRs to reduce developer effort. Despite this positive impact, 30.7% of AI-generated suggestions during code reviews were still met with skepticism or disagreement [41]. Our study fills this gap by tracing 567 autonomous PRs created by Claude Code across 157 open-source projects, reporting acceptance rates, revision effort, and rejection rationales.

Researchers have proposed various approaches for directly involving LLMs in activities related to issue reports and PRs. For example, Kang *et al.* [21] used LLMs to generate test cases from issue reports for easier bug reproduction, Zhang *et al.* [19] trained LLMs to automatically generate PR titles, and Bo *et al.* [3] leveraged ChatGPT to improve the clarity of issue reports for helping subsequent tasks such as bug fixing.

Complementary work investigates AI agents that clarify ambiguous requirements and collaboratively co-evolve PR descriptions; for instance, AgileGen aligns end-user perspectives with acceptance criteria [46], while large-scale analyses of LLMs on code-change tasks reveal their potential for automated review of PR diffs [10]. Multi-agent benchmarks also examine ChatGPT's ability to self-verify its generated code through test-report generation, reducing reviewer burden [44].

Early work by Tufano *et al.* [39] pioneered DL models that propose code changes before review submission and apply reviewer suggestions after submission. Guo *et al.* report that GPT-4 can refine 66% of files given review comments [12], and later boost intent-extraction accuracy to 79% in their ICSE '25 study [13]. To improve developer confidence, Di and Zhang [8] interleave inline comments with LLM edits, cutting task time by 16.7 percent.

[46] https://openai.com/index/introducing-codex

6.3 Empirical studies of automatic code generation

Researchers have studied various aspects of AI-generated code, including correctness [2], efficiency [31], and security [26]. For example, Billah *et al.* [2] evaluated the correctness (pass rate) of LLMs in solving programming challenges on competitive programming platforms such as LeetCode and Codeforces. They reported a decrease in the pass rate in live Codeforces contests compared to archived problems. However, code correctness alone does not necessarily imply efficiency or optimality as code complexity increases, and extra effort is required to support AI models in generating efficient code [31]. From a security perspective, Liu *et al.* [26] identified security vulnerabilities in AI-generated code and highlighted potential security risks that must be mitigated before deployment in production environments. Beyond generating code at the granularity of individual functions or files, SWE-bench [20] was introduced to evaluate the performance of AI-powered code generation tools in resolving real-world GitHub issues.

Subsequent work has shown that ChatGPT often produces buggy or low-quality code that benefits from iterative repair [26], while self-collaboration frameworks with multiple LLM agents significantly improve correctness and reduce human intervention [9]. Empirical studies highlight the challenges of domain-specific coding (*e.g.,* machine-learning pipelines) and demonstrate that knowledge-prompting strategies boost performance in such settings [11, 37]. Recent surveys map out future research directions for automatic programming pipelines, including integrated quality-assurance stages [27], and bias-testing frameworks reveal that LLMs can inherit societal biases, motivating prompt-engineering mitigation strategies [18].

Wang *et al.* [40] catalogue 557 distinct error patterns that six LLMs make on HumanEval. Niu *et al.* benchmark 19 pre-trained models across 13 tasks, confirming that functional correctness rarely correlates with efficiency [30]. O'Brien *et al.* [32] show Copilot frequently injects TODO-style self-admitted technical debt. For repair, Xia *et al.* [43] and Bouzenia *et al.* [4] re-cast bug fixing as an autonomous planning problem executed by an LLM agent.

7 Threats to Validity

7.1 Threats to internal validity

A potential concern in our study is the ambiguity related to developers' behaviors when using Claude Code. Developers may modify the generated code before committing it. Conversely, even when developers use Claude Code to generate code, they may directly push the changes without requesting the tool to perform the commit itself. To mitigate this threat, we explicitly included only PRs clearly marked as "Generated with Claude Code." This explicit labeling provides a strong signal of Claude Code's involvement, reducing potential ambiguity about code provenance.

7.2 Threats to construct validity

Claude Code implements agentic coding and can leverage the Model Context Protocol (MCP), which enables agents to integrate with external tools and data sources. However, this study does not isolate the specific contributions of agentic capabilities and MCP integration to the observed outcomes. Disentangling their individual impacts remains an important direction for future research.

7.3 Threats to external validity

This study examines only 567 PRs that explicitly used Claude Code, which represents the full set we were able to retrieve. One reason for this limited sample size is that Claude Code was launched only recently, in February 2025. To validate our findings, replication studies with larger datasets collected over a longer time frame will be necessary. Furthermore, our analysis focused exclusively on Claude Code, an agentic coding tool whose usage could be reliably identified in PRs. As a result,

our findings may not generalize to other agentic coding tools. Still, we believe that the findings here may be indicative of broader trends, given that these functionalities and performance are similar.

8 Conclusion

This paper presents the first empirical study investigating the impact of agentic coding tools, specifically Claude Code, on open-source projects. We analyzed 567 pull requests across 157 open-source projects to understand how these AI-generated contributions are received by developers. Our findings show that while Agentic-PRs are accepted at a lower rate than Human-PRs (83.8% vs. 91.0%), they are still widely adopted in real-world projects. Importantly, when revisions are required, the extent of modifications does not differ significantly between Agentic-PRs and Human-PRs. This suggests that once reviewers engage with Agentic-PRs, the additional effort needed to integrate them is similar to that required for human contributions. Overall, agentic coding provides a strong starting point that benefits from human oversight to ensure correctness, maintainability, and alignment with project conventions. In practice, developers can reduce review friction by keeping PRs small and encoding project-specific rules and architecture guidance for agents, while agent builders can integrate project-aware CI/CD checks and support automated maintenance tasks such as rebasing and conflict resolution.

Acknowledgments

We gratefully acknowledge the financial support of JSPS KAKENHI grants (JP24K02921, JP25K21359), as well as JST PRESTO grant (JPMJPR22P3), ASPIRE grant (JPMJAP2415), and AIP Accelerated Program (JPMJCR25U7).

References

[1] Adam Alami, Victor Vadmand Jensen, and Neil A. Ernst. 2025. Accountability in Code Review: The Role of Intrinsic Drivers and the Impact of LLMs. *ACM Trans. Softw. Eng. Methodol.* (2025).

[2] Md Mustakim Billah, Palash Ranjan Roy, Zadia Codabux, and Banani Roy. 2024. Are Large Language Models a Threat to Programming Platforms? An Exploratory Study. In *Proceedings of the 18th IEEE/ACM International Symposium on Empirical Software Engineering and Measurement (ESEM'24)*. 292–301. https://doi.org/10.1145/3674805.3686689

[3] Lili Bo, Wangjie Ji, Xiaobing Sun, Ting Zhang, Xiaoxue Wu, and Ying Wei. 2024. ChatBR: Automated assessment and improvement of bug report quality using ChatGPT. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering (ASE'24)*. 1472–1483. https://doi.org/10.1145/3691620.3695518

[4] Islem Bouzenia, Premkumar T. Devanbu, and Michael Pradel. 2025. RepairAgent: An Autonomous, LLM-Based Agent for Program Repair. In *Proceedings of the 47th IEEE/ACM International Conference on Software Engineering (ICSE'25)*. 2188–2200.

[5] Moataz Chouchen, Narjes Bessghaier, Mahi Begoug, Ali Ouni, Eman Abdullah AlOmar, and Mohamed Wiem Mkaouer. 2024. How Do So ware Developers Use ChatGPT? An Exploratory Study on GitHub Pull Requests. In *Proceedings of the 21st IEEE/ACM International Conference on Mining Software Repositories (MSR'24)*. 212–216. https://doi.org/10.1145/3643991.3645084

[6] Mariana Coutinho, Lorena Marques, Anderson Santos, Márcio Dahia, César França, and Ronnie de Souza Santos. 2024. The Role of Generative AI in Software Development Productivity: A Pilot Case Study. In *Proceedings of the 1st IEEE/ACM International Conference on AI-Powered Software (AIware'24)*. 15–16. https://doi.org/10.1145/3664646.3664773

[7] Arghavan Moradi Dakhel, Vahid Majdinasab, Amin Nikanjam, Foutse Khomh, Michel C. Desmarais, and Zhen Ming (Jack) Jiang. 2023. GitHub Copilot AI pair programmer: Asset or Liability? *J. Syst. Softw.* 203 (2023), 111734. https://doi.org/10.1016/J.JSS.2023.111734

[8] Yifeng Di and Tianyi Zhang. 2025. Enhancing Code Generation via Bidirectional Comment-Level Mutual Grounding. In *Proceedings of the 47th IEEE/ACM International Conference on Software Engineering (ICSE'25)*. 1359–1371.

[9] Yihong Dong, Xue Jiang, Zhi Jin, and Ge Li. 2024. Self-Collaboration Code Generation via ChatGPT. *ACM Trans. Softw. Eng. Methodol.* 33, 7 (2024), 189:1–189:38.

[10] Lishui Fan, Jiakun Liu, Zhongxin Liu, David Lo, Xin Xia, and Shanping Li. 2025. Exploring the Capabilities of LLMs for Code-Change-Related Tasks. *ACM Trans. Softw. Eng. Methodol.* 34, 6 (2025), 159:1–159:36.

[11] Xiaodong Gu, Meng Chen, Yalan Lin, Yuhan Hu, Hongyu Zhang, Chengcheng Wan, Zhao Wei, Yong Xu, and Juhong Wang. 2025. On the Effectiveness of Large Language Models in Domain-Specific Code Generation. *ACM Trans. Softw. Eng. Methodol.* 34, 3 (2025), 78:1–78:22.

[12] Qi Guo, Junming Cao, Xiaofei Xie, Shangqing Liu, Xiaohong Li, Bihuan Chen, and Xin Peng. 2024. Exploring the Potential of ChatGPT in Automated Code Refinement: An Empirical Study. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering (ICSE'24)*. 34:1–34:13.

[13] Qi Guo, Xiaofei Xie, Shangqing Liu, Ming Hu, Xiaohong Li, and Lei Bu. 2025. Intention is All you Need: Refining your Code from your Intention. In *Proceedings of the 47th IEEE/ACM International Conference on Software Engineering (ICSE'25)*. 1127–1139.

[14] Mohammed Mehedi Hasan, Hao Li, Emad Fallahzadeh, Gopi Krishnan Rajbahadur, Bram Adams, and Ahmed E. Hassan. 2025. Model Context Protocol (MCP) at First Glance: Studying the Security and Maintainability of MCP Servers. *CoRR* abs/2506.13538 (2025).

[15] Ahmed E. Hassan, Gustavo Ansaldi Oliva, Dayi Lin, Boyuan Chen, and Zhen Ming Jiang. 2024. Towards AI-Native Software Engineering (SE 3.0): A Vision and a Challenge Roadmap. *CoRR* abs/2410.06107 (2024). https://doi.org/10.48550/ARXIV.2410.06107

[16] Junda He, Christoph Treude, and David Lo. 2025. LLM-Based Multi-Agent Systems for Software Engineering: Literature Review, Vision, and the Road Ahead. *ACM Trans. Softw. Eng. Methodol.* 34, 5 (2025), 124:1–124:30. https://doi.org/10.1145/3712003

[17] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2024. Large Language Models for Software Engineering: A Systematic Literature Review. *ACM Trans. Softw. Eng. Methodol.* 33, 8 (2024), 220:1–220:79. https://doi.org/10.1145/3695988

[18] Dong Huang, Jie M. Zhang, Qingwen Bu, Xiaofei Xie, Junjie Chen, and Heming Cui. 2025. Bias Testing and Mitigation in LLM-based Code Generation. *ACM Trans. Softw. Eng. Methodol.* (2025).

[19] Ivana Clairine Irsan, Ting Zhang, Ferdian Thung, David Lo, and Lingxiao Jiang. 2022. AutoPRTitle: A Tool for Automatic Pull Request Title Generation. In *Proceedings of the 38th IEEE International Conference on Software Maintenance and Evolution (ICSME'22)*. 454–458. https://doi.org/10.1109/ICSME55016.2022.00058

[20] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. 2024. SWE-bench: Can Language Models Resolve Real-world Github Issues?. In *Proceedings of the 12th International Conference on Learning Representations (ICLR'24)*. 7–11.

[21] Sungmin Kang, Juyeon Yoon, and Shin Yoo. 2023. Large Language Models are Few-shot Testers: Exploring LLM-based General Bug Reproduction. In *Proceedings of the 45th IEEE/ACM International Conference on Software Engineering, (ICSE'23)*. 2312–2323. https://doi.org/10.1109/ICSE48619.2023.00194

[22] Stefano Lambiase, Gemma Catolino, Fabio Palomba, and Filomena Ferrucci. 2025. Motivations, Challenges, Best Practices, and Benefits for Bots and Conversational Agents in Software Engineering: A Multivocal Literature Review. *ACM Comput. Surv.* 57, 4 (2025), 93:1–93:37. https://doi.org/10.1145/3704806

[23] J. Richard Landis and Gary G. Koch. 1977. The Measurement of Observer Agreement for Categorical Data. *Biometrics* 33 (1977), 159–174.

[24] Feng Lin, Dong Jae Kim, and Tse-Hsun Chen. 2025. SOEN-101: Code Generation by Emulating Software Process Models Using Large Language Model Agents. In *Proceedings of the 47th IEEE/ACM International Conference on Software Engineering (ICSE'25)*. 1527–1539.

[25] Chao Liu, Xindong Zhang, Hongyu Zhang, Zhiyuan Wan, Zhan Huang, and Meng Yan. 2024. An Empirical Study of Code Search in Intelligent Coding Assistant: Perceptions, Expectations, and Directions. In *Proceedings of the 32th IEEE/ACM International Conference on the Foundations of Software Engineering (FSE'24)*. 283–293. https://doi.org/10.1145/3663529.3663848

[26] Zhijie Liu, Yutian Tang, Xiapu Luo, Yuming Zhou, and Liang Feng Zhang. 2024. No Need to Lift a Finger Anymore? Assessing the Quality of Code Generation by ChatGPT. *IEEE Trans. Software Eng.* 50, 6 (2024), 1548–1584. https://doi.org/10.1109/TSE.2024.3392499

[27] Michael R. Lyu, Baishakhi Ray, Abhik Roychoudhury, Shin Hwei Tan, and Patanamon Thongtanunam. 2025. Automatic Programming: Large Language Models and Beyond. *ACM Trans. Softw. Eng. Methodol.* 34, 5 (2025), 140:1–140:33.

[28] Moritz Mock, Jorge Melegati, and Barbara Russo. 2024. Generative AI for Test Driven Development: Preliminary Results. In *Proceeding of the 25th Software Engineering and Extreme Programming Workshops (XP'24)*. 24–32. https://doi.org/10.1007/978-3-031-72781-8_3

[29] Nadia Nahar, Haoran Zhang, Grace A. Lewis, Shurui Zhou, and Christian Kästner. 2025. The Product Beyond the Model - An Empirical Study of Repositories of Open-Source ML Products. In *Proceedings of the 47th IEEE/ACM International Conference on Software Engineering (ICSE'25)*. 1540–1552.

[30] Changan Niu, Chuanyi Li, Vincent Ng, Dongxiao Chen, Jidong Ge, and Bin Luo. 2023. An Empirical Comparison of Pre-Trained Models of Source Code. In *Proceedings of the 45th IEEE/ACM International Conference on Software*

Engineering (ICSE'23). 2136–2148.

[31] Changan Niu, Ting Zhang, Chuanyi Li, Bin Luo, and Vincent Ng. 2024. On Evaluating the Efficiency of Source Code Generated by LLMs. In *Proceedings of the 1th IEEE/ACM International Conference on AI Foundation Models and Software Engineering (FORGE'24)*. 103–107. https://doi.org/10.1145/3650105.3652295

[32] David O'Brien, Sumon Biswas, Sayem Mohammad Imtiaz, Rabe Abdalkareem, Emad Shihab, and Hridesh Rajan. 2024. Are Prompt Engineering and TODO Comments Friends or Foes? An Evaluation on GitHub Copilot. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering (ICSE'24)*. 219:1–219:13.

[33] Jevgenija Pantiuchina, Bin Lin, Fiorella Zampetti, Massimiliano Di Penta, Michele Lanza, and Gabriele Bavota. 2022. Why Do Developers Reject Refactorings in Open-Source Projects? *ACM Trans. Softw. Eng. Methodol.* 31, 2 (2022), 23:1–23:23.

[34] David Piorkowski, Soya Park, April Yi Wang, Dakuo Wang, Michael J. Muller, and Felix Portnoy. 2021. How AI Developers Overcome Communication Challenges in a Multidisciplinary Team: A Case Study. *Proc. ACM Hum. Comput. Interact.* 5 (2021), 131:1–131:25.

[35] Tajmilur Rahman, Yuecai Zhu, Lamyea Maha, Chanchal Roy, Banani Roy, and Kevin A. Schneider. 2024. Take Loads Off Your Developers: Automated User Story Generation using Large Language Model. In *Proceedings of the 40th IEEE International Conference on Software Maintenance and Evolution (ICSME'24)*. 791–801. https://doi.org/10.1109/ICSME58944.2024.00082

[36] Robson Santos, Ítalo Santos, Cleyton V. C. de Magalhães, and Ronnie de Souza Santos. 2024. Are We Testing or Being Tested? Exploring the Practical Applications of Large Language Models in Software Testing. In *Proceedings of 17th IEEE International Conference on Software Testing, Verification and Validation (ICST'24)*. 353–360. https://doi.org/10.1109/ICST60714.2024.00039

[37] Jiho Shin, Moshi Wei, Junjie Wang, Lin Shi, and Song Wang. 2024. The Good, the Bad, and the Missing: Neural Code Generation for Machine Learning Tasks. *ACM Trans. Softw. Eng. Methodol.* 33, 2 (2024), 51:1–51:24.

[38] Rosalia Tufano, Antonio Mastropaolo, Federica Pepe, Ozren Dabic, Massimiliano Di Penta, and Gabriele Bavota. 2024. Unveiling ChatGPT's Usage in Open Source Projects: A Mining-based Study. In *Proceedings of the 21st IEEE/ACM International Conference on Mining Software Repositories (MSR'24)*. 571–583.

[39] Rosalia Tufano, Luca Pascarella, Michele Tufano, Denys Poshyvanyk, and Gabriele Bavota. 2021. Towards Automating Code Review Activities. In *Proceedings of the 43rd IEEE/ACM International Conference on Software Engineering (ICSE'21)*. 163–174.

[40] Zhijie Wang, Zijie Zhou, Da Song, Yuheng Huang, Shengmai Chen, Lei Ma, and Tianyi Zhang. 2025. Towards Understanding the Characteristics of Code Generation Errors Made by Large Language Models. In *Proceedings of the 47th IEEE/ACM International Conference on Software Engineering (ICSE'25)*. 2587–2599.

[41] Miku Watanabe, Yutaro Kashiwa, Bin Lin, Toshiki Hirao, Ken-ichi Yamaguchi, and Hajimu Iida. 2024. On the Use of ChatGPT for Code Review: Do Developers Like Reviews By ChatGPT?. In *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering (EASE'24)*. 375–380. https://doi.org/10.1145/3661167.3661183

[42] Peter Weißgerber, Daniel Neu, and Stephan Diehl. 2008. Small patches get in!. In *Proceedings of the 2008 International Working Conference on Mining Software Repositories (MSR'08)*. 67–76.

[43] Chunqiu Steven Xia, Yuxiang Wei, and Lingming Zhang. 2023. Automated Program Repair in the Era of Large Pre-trained Language Models. In *Proceedings of the 45th IEEE/ACM International Conference on Software Engineering (ICSE'23)*. 1482–1494.

[44] Xiao Yu, Lei Liu, Xing Hu, Jacky Wai Keung, Jin Liu, and Xin Xia. 2024. Fight Fire With Fire: How Much Can We Trust ChatGPT on Source Code-Related Tasks? *IEEE Trans. Software Eng.* 50, 12 (2024), 3435–3453. https://doi.org/10.1109/TSE.2024.3492204

[45] Qunhong Zeng, Yuxia Zhang, Zhiqing Qiu, and Hui Liu. 2025. A First Look at Conventional Commits Classification. In *Proceedings of the 47th IEEE/ACM International Conference on Software Engineering (ICSE'25)*. 2277–2289.

[46] Sai Zhang, Zhenchang Xing, Ronghui Guo, Fangzhou Xu, Lei Chen, Zhaoyuan Zhang, Xiaowang Zhang, Zhiyong Feng, and Zhiqiang Zhuang. 2025. Empowering Agile-Based Generative Software Development through Human-AI Teamwork. *ACM Trans. Softw. Eng. Methodol.* 34, 6 (2025), 156:1–156:46.