

Where Do AI Coding Agents Fail? An Empirical Study of Failed Agentic Pull Requests in GitHub

Ramtin Ehsani
Drexel University
Philadelphia, PA, USA
ramtin.ehsani@drexel.edu

Abdullah Al Mujahid
Missouri University of Science and Technology
Rolla, MO, USA
amgzc@mst.edu

Sakshi Pathak
Drexel University
Philadelphia, PA, USA
sp3856@drexel.edu

Mia Mohammad Imran
Missouri University of Science and Technology
Rolla, MO, USA
imranm@mst.edu

Shriya Rawal
Drexel University
Philadelphia, PA, USA
sr3728@drexel.edu

Preetha Chatterjee
Drexel University
Philadelphia, PA, USA
preetha.chatterjee@drexel.edu

Abstract

AI coding agents are now submitting pull requests (PRs) to software projects, acting not just as assistants but as autonomous contributors. As these agentic contributions are rapidly increasing across real repositories, little is known about how they behave in practice and why many of them fail to be merged. In this paper, we conduct a large-scale study of 33k agent-authored PRs made by five coding agents across GitHub. **(RQ1)** We first quantitatively characterize merged and not-merged PRs along four broad dimensions: 1) merge outcomes across task types, 2) code changes, 3) CI build results, and 4) review dynamics. We observe that tasks related to *documentation*, *CI*, and *build update* achieve the highest merge success, whereas *performance* and *bug-fix* tasks perform the worst. Not-merged PRs tend to involve larger code changes, touch more files, and often do not pass the project's CI/CD pipeline validation. **(RQ2)** To further investigate why some agentic PRs are not merged, we qualitatively analyze 600 PRs to derive a hierarchical taxonomy of rejection patterns. This analysis complements the quantitative findings in RQ1 by uncovering rejection reasons not captured by quantitative metrics, including *lack of meaningful reviewer engagement*, *duplicate PRs*, *unwanted feature implementations*, and *agent misalignment*. Together, our findings highlight key socio-technical and human-AI collaboration factors that are critical to improving the success of future agentic workflows.

CCS Concepts

- Software and its engineering → Software creation and management;
- Computing methodologies → Intelligent agents.

Keywords

Agents, Large language models, Agentic pull request, AIDev

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

MSR 2026, Rio de Janeiro, Brazil

© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-XXXX-X/2018/06
<https://doi.org/XXXXXXX.XXXXXXX>

ACM Reference Format:

Ramtin Ehsani, Sakshi Pathak, Shriya Rawal, Abdullah Al Mujahid, Mia Mohammad Imran, and Preetha Chatterjee. 2018. Where Do AI Coding Agents Fail? An Empirical Study of Failed Agentic Pull Requests in GitHub. In *Proceedings of MSR '26: Proceedings of the 23rd International Conference on Mining Software Repositories (MSR 2026)*. ACM, New York, NY, USA, 5 pages. <https://doi.org/XXXXXXX.XXXXXXX>

1 Introduction

AI coding agents such as GitHub Copilot and OpenAI Codex are rapidly becoming active contributors to open-source repositories, often assisting with or directly authoring new pull requests (PRs). Beyond offering inline code suggestions, these tools now generate code changes, respond to reviewer feedback, and participate in the software lifecycle as autonomous agents [2, 4, 9, 26, 40]. As agent-authored PRs are becoming more prevalent, it is critical to understand how they are evaluated and accepted in practice.

Prior work shows that PR acceptance depends on factors such as technical correctness, problem scope, and contributor reputation [25, 35, 42]. PRs are more likely to be merged when they pass tests and CI pipelines, address high-priority or well-scoped problems, and introduce localized and incremental code changes rather than broad or invasive modifications [38, 41, 43]. While these factors characterize the success of human-authored PRs, their relevance and applicability to agent-authored PRs are not yet well understood.

Coding agents have been extensively benchmarked across a range of tasks, from code generation [5, 33], testing [23, 31, 39], to automated program repair [8, 22, 30]. Other studies have analyzed agent-driven code refactoring, reporting that these refactorings tend to be small, localized improvements that produce modest but statistically significant gains in code quality [21, 34]. More recent work has focused on agent reasoning and execution behaviors, including traceability, decision-making, and workflow strategies in complex software engineering tasks [3, 28]. While prior work evaluates agents in isolated tasks, we lack a systematic assessment of how agents perform when integrated into real development workflows involving CI validation, code review, and iterative revision.

In this paper, we conduct a large-scale empirical study on agent-authored pull requests using the AIDev-pop dataset [26], which comprises over 33k PRs submitted by five major coding agents across GitHub projects with more than 100 stars. We characterize the types of contributions agents attempt, their acceptance rates,

reviewer interactions, and, most importantly, where and why their contributions fail. Specifically, we investigate two RQs:

RQ1: How do merged and not-merged agent-authored PRs differ in task types, code changes, CI outcomes, and review interactions? We find that agentic PRs involving *documentation*, *CI*, and *build update* tasks are merged at higher rates, while *performance* and *bug-fix* contributions show the lowest acceptance. Not-merged PRs tend to involve larger code changes, touch more files, receive more reviewer revisions, and frequently fail project CI checks.

RQ2: What patterns lead to agent-authored PRs not being merged in real-world software repositories? The most frequent rejection pattern is reviewer abandonment, where agent-authored PRs receive little or no human engagement before being closed. Among PRs that do undergo active review, duplicate PRs, build failures, and unwanted features account for the majority of rejections.

Overall, our results suggest that agentic PR failures stem from misalignment with repository workflows (e.g., CI/CD failures), developer expectations (e.g., unwanted or incorrect features), and a lack of project coordination (e.g., reviewer abandonment).

2 Methodology

RQ1: How do merged and not-merged agent-authored PRs differ in task types, code changes, CI outcomes, and review interactions?

We perform a quantitative characterization of agent-authored pull requests along four dimensions: 1) merge outcomes across task types, 2) code changes, 3) CI build results, and 4) review dynamics. These characteristics are grounded in prior work on factors that influence pull request acceptance [25, 41, 42].

We examine **merge outcomes across task types**, using the task labels provided in the dataset. These tasks consist of 11 categories: feature, fix, performance, refactoring, style, documentation, test, chore, build, CI, and other [7, 26]. This allows us to assess whether certain categories of agent-generated contributions are more or less likely to be merged. We analyze the magnitude of proposed **code changes** by measuring a) the total number of added and removed lines of code (*#LOC Changes*), and b) the number of files modified by each PR (*#File Changes*). These two characteristics serve as quantitative indicators of the PR's complexity and potential review burden [25, 41, 42]. We inspect **CI build results** for each PR by querying the GitHub API for the status of the last commit in the PR. For every PR, we extract the number of failed check-runs (*#Failed CI Checks*) and record the overall commit status reported by GitHub (success or failure). This provides a proxy for whether the agent-generated changes break tests, violate linting rules, or fail other repository-specific validation pipelines. This metric allows us to capture automated quality signals that may influence maintainers' decisions [27, 41]. We examine **review interactions** associated with each PR [41]. Specifically, we compute a) the number of *review comments* in a PR (*#Review Comments*), and b) the number of *review revisions* each PR receives (*#Review Revisions*). Review revisions are the total number of additions and deletions by the developers during review cycles of the PRs. These measures reflect how much developer attention and iteration an agent-generated PR demands.

Because of the dataset size ($> 33k$), standard statistical significance testing by itself is not informative because all comparisons might yield statistically significant values even when the differences

are negligible [36]. Instead, following best practices in large-scale empirical studies [20, 24], we rely on effect size measures, using Cliff's delta (δ) to quantify the magnitude of difference between merged and not-merged PR distributions. To complement effect-size analysis, we use kernel density estimates [20, 24] to visualize distribution shapes. Unlike simple summaries, kernel density plots give a smooth, continuous view of the data's distribution, making it easier to see shifts and spread in the data [24, 37]. In addition, we use logistic regression modeling [24] to see how effective these characteristics are in predicting the outcome of agent-authored PRs. Together, these analyses allow us to assess whether meaningful differences exist between merged and not-merged agentic PRs.

RQ2: What patterns lead to agent-authored PRs not being merged in real-world software repositories?

We randomly select a subset of 600 rejected PRs for qualitative analysis, stratified across the five coding agents to ensure balanced coverage, and sufficient statistical power to estimate rejection prevalence with 95% confidence and a $\pm 5\%$ margin of error [6].

We start with a sample of 100 rejected PRs. Following open coding [1], two authors independently label each PR with its primary reason for rejection. During the first round of coding (50 PRs), the annotators identified recurring patterns such as build failures, licensing or contribution policy violations, redundant or unwanted changes, and logical or semantic errors in the proposed code. These patterns were iteratively discussed to derive an initial hierarchical taxonomy spanning three levels: **CODE**, **PULL REQUEST**, and **REVIEWER** level. We assess inter-rater reliability using Cohen's kappa [29]. After the first round, the agreement was 0.55, indicating moderate alignment. All disagreements were then discussed and resolved through group discussions, during which the taxonomy was refined, and an additional **AGENTIC** level was introduced. The refined taxonomy was applied to a second set of 50 PRs, resulting in a final Cohen's kappa of 0.91 across all 100 PRs, reflecting strong agreement [29]. Using this taxonomy, the annotators independently labeled 250 more PRs each (500 in total), bringing the size of the manually annotated dataset to 600 PRs. The detailed annotation instructions and codebook are available in the replication package.

Our iterative manual coding resulted in *a hierarchical taxonomy of agentic-PR rejection patterns*, consisting of four high-level categories as described in Table 2. 1) **REVIEWER** level reflects PRs that close without meaningful developer engagement, often due to inactivity or abandonment. 2) **PULL REQUEST** level capture cases where the PR is unsuitable for project integration, such as unwanted features, duplicate submissions, or changes submitted to the wrong branch. 3) **CODE** level failures happen when the proposed implementation is incomplete, incorrect, or breaks CI/test pipelines. 4) **AGENTIC** failures capture behaviors such as licensing violations with the project or misalignment with reviewer instructions.

3 Results

RQ1: How do merged and not-merged agent-authored PRs differ in task types, code changes, CI outcomes, and review interactions?

Of the 33,596 agentic pull requests, the majority originate from OpenAI Codex (21,799), exceeding the output of any other agent by more than a factor of four. Copilot and Devin follow with 4,970 and 4,827 PRs, while Cursor contributes 1,541, and Claude Code

represents the smallest share with 459 PRs. Across all agents, 71.48% of PRs (24,014) are successfully merged. Despite handling the largest volume of contributions, OpenAI Codex achieves the highest merge rate, with 82.59% (18,004) of its PRs being accepted. Cursor follows with a 65.22% merge rate (1,005), then Claude Code at 59.04% (271), and Devin at 53.76% (2,595). Copilot exhibits the lowest merge rate, with only 43.04% (2,139) of its PRs being merged.

	Feat	Fix	Docs	Test	Refactor	Chore	Build	Ci	Perf	Style	Revert
Codex	0.57 n:250	0.57 n:115	0.75 n:32	0.50 n:6	0.50 n:26	0.71 n:14	0.88 n:8	0.80 n:5	0.67 n:3	0 n:0	0 n:0
Copilot	0.38 n:1661	0.42 n:1993	0.61 n:458	0.37 n:167	0.48 n:301	0.44 n:126	0.50 n:121	0.63 n:67	0.27 n:44	0.40 n:15	0.80 n:5
Cursor	0.60 n:616	0.68 n:411	0.74 n:207	0.62 n:37	0.65 n:111	0.74 n:50	0.57 n:42	0.94 n:16	0.46 n:24	0.72 n:18	0 n:0
Devin	0.54 n:1904	0.43 n:1249	0.71 n:620	0.50 n:117	0.57 n:437	0.57 n:281	0.58 n:64	0.61 n:59	0.35 n:62	0.68 n:25	0.67 n:6
Claude	0.81 n:10019	0.82 n:4338	0.92 n:2570	0.84 n:2029	0.80 n:1413	0.84 n:425	0.87 n:392	0.86 n:264	0.68 n:207	0.85 n:130	0.80 n:5

Figure 1: Merge-rate per Task Type Across Agentic PRs.

Figure 1 shows the merge rates for each agent across all **task categories**, along with the number of PRs in each cell. The distribution of success varies across task types and agents. Codex shows the highest merge rates overall, exceeding 80% in categories including *documentations* (0.92), *CI* (0.86), *build* (0.87), *chore* (0.84), *test* (0.84), *fix* (0.82), *feature* (0.81), and *refactoring* (0.80). Its lowest category is *performance* (0.68). Cursor also performs strongly on maintenance-oriented tasks, with high merge rates in *CI* (0.94), *documentations* (0.74), *chore* (0.74), and *style* (0.72). Its lowest rates occur in *build* (0.57) and *performance* (0.46). Claude Code’s highest merge rates appear in *build* (0.88), *documentations* (0.75), and *CI* (0.57), with lower rates for *test* (0.50) and *refactoring* (0.50). Devin shows moderate values across most categories, with higher merge rates in *documentations* (0.71), *style* (0.68), and *CI* (0.61), and lower rates in *performance* (0.35) and *fix* (0.43). Copilot displays the lowest merge rates among the agents, with its highest values in *CI* (0.63) and *documentations* (0.61), and its lowest in *feature* (0.38), *test* (0.37), and *performance* (0.27). Across agents, tasks with consistently higher merge rates include **documentations** (84%), **CI** (79%), and **build** (74%). In contrast, **performance** (55%) and **fix** (64%) display the lowest merge rates overall. These results indicate that tasks involving documentation, CI, and build changes tend to merge more easily in repositories, whereas categories requiring more complex or subjective changes show lower merge rates.

Figures 2a and 2b show the differences in **code changes** between merged and not-merged PRs, in terms of *#LOC Changes* and *#File Changes*. Not merged PRs tend to introduce larger modifications in the number of files and LOC than merged ones. Based on Cliff’s δ (Table 1), the difference in total lines of code changes is 17%, and the difference in the number of changed files is 10%, both indicating a small-to-medium effect size that not-merged PRs skew toward larger changes. Figures 2a and 2b also show kernel density estimates of these two metrics. Because changes span several orders of magnitude, the distributions for these plots are shown on a log10 scale, where each unit corresponds to a ten-fold increase in size. Higher density toward the right side of the plots reflects larger PRs. In both subfigures, the distribution for not-merged PRs is shifted slightly rightward relative to merged PRs, visualizing the same trend captured by Cliff’s δ .

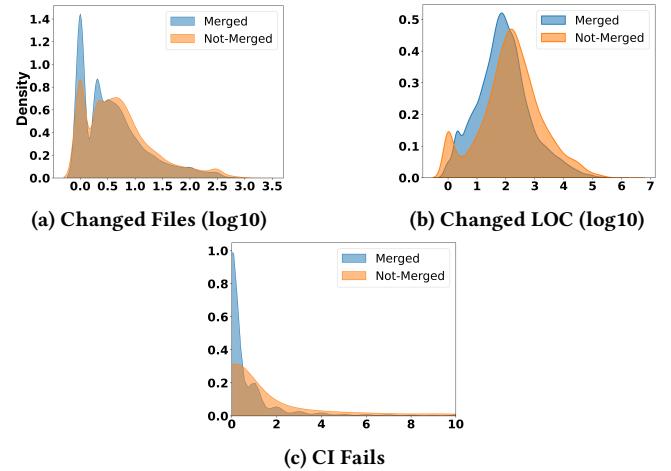


Figure 2: Differences in Merged vs. Not-merged PRs.

Figure 2c reports the distribution of **CI failures** for merged and not-merged agentic PRs. Not-merged PRs show a noticeably heavier tail, with many PRs accumulating multiple failing check runs. In contrast, merged PRs cluster sharply near zero, indicating that most merged contributions pass their checks with little or no failure. Cliff’s δ of 24% indicates a moderate effect size, showing that not-merged PRs tend to experience more CI failures.

Table 1: Logistic Regression and Effect Size (* $p < 0.05$)

Characteristic	Coef	p-value	Odds Ratio	δ
#LOC Changes	-2.8e-06	~1%*	99%	-0.17
#File Changes	-0.0011	~1%*	99%	-0.10
#Failed CI Checks	-0.1579	~1%*	85%	-0.24
#Review Comments	-0.0028	~48%	99%	-0.05
#Review Revisions	-1.6e-05	~67%	99%	-0.03

Figure 3 shows the differences in **review interactions**. Not-merged PRs tend to receive more *reviewer revisions* than merged ones. Based on Cliff’s δ (Table 1), developers make approximately 5% more *review comments* and 3% more *revisions* on not-merged PRs compared to merged PRs, although the effect sizes for both are small. The density distributions follow a similar overall pattern. However, the curves widen for not-merged PRs as the number of comments (Figure 3a) or revisions (Figure 3b) increases, indicating that not-merged PRs often undergo extensive reviewer discussion and iterative refinement until a final decision is made to leave them open or mark them as rejected.

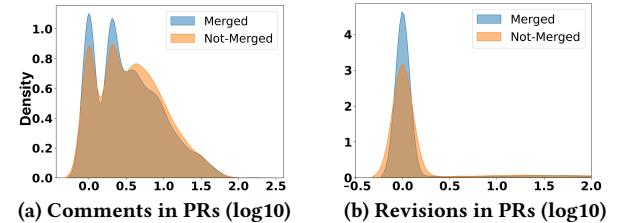


Figure 3: Reviews of Merged vs. Not-merged PRs.

The logistic regression results in Table 1 show that all model coefficients are negative, indicating that increases in these metrics are associated with lower odds of a PR being merged. Based on the *p*-values, all variables except *#review comments* and *#review revisions* are significant (threshold at 0.05). Odds ratios quantify the practical effect of each metric. For example, a one-unit increase in

#LOC changes decreases the odds of a merge by approximately 1%, which can accumulate meaningfully given that code modifications span several orders of magnitude in size. Similar patterns hold for #file changes, #review comments, and #revision cycles, although with smaller magnitudes. Notably, each additional failed CI check decreases the odds of a merge by about 15%.

Across all agentic PRs, *documentation*, *CI*, and *build* tasks exhibit the highest success, while *performance* and *fix* see the lowest. Not-merged PRs tend to introduce larger code changes, touch more files, fail more CI checks, and receive slightly more review comments and revisions.

RQ2: What patterns lead to agent-authored PRs not being merged in real-world software repositories?

Table 2 summarizes the distribution of rejection patterns for the 600 manually annotated rejected agentic pull requests. We note that 38 PRs were no longer accessible at the time of analysis due to deletion or archival, leaving a total of 562 PRs for categorization.

REVIEWER-level abandonment is the most frequent rejection pattern, accounting for 228 PRs (38%). These PRs were left without any meaningful human reviewer interaction, often after prolonged inactivity or automated closure, indicating that a substantial fraction of agentic PRs fail before entering active review.

PULL REQUEST-level reasons form the second-largest group, comprising 188 PRs (31%). Within this level, duplicate PRs are the most common pattern, affecting 142 PRs (23%), where maintainers explicitly reference an existing PR that already implements the same change. For example, one PR was closed with the comment: “*Superseded by PR #715 which consolidates all GFQL code changes into a single PR*” [10]. Unwanted features account for 24 PRs (4%), where maintainers state that the contribution is misaligned with project goals or introduces excessive or unnecessary changes. Examples include: “*Too old already superseded by more recent pushes*” [11] and “*This is a LOT to review, would really prefer smaller granular PRs*” [12]. Less frequent patterns include non-functional PRs (13; 2%), which often consist of only setup or configuration tests. For example, PRs explicitly titled “*testing DO NOT MERGE*” [13]. Wrong task descriptions account for 7 PRs (1%), where the PR description provides little to no meaningful context. Maintainers often respond with comments such as “*Sorry, I don’t know what this is, but it doesn’t look like it belongs in our repo*” [14]. Finally, wrong branch submissions (2; <1%) occur when PRs are opened on incorrect branches, prompting maintainers to make comments such as “*PR is opened against main. You probably want to open it against develop*” [15].

CODE-level reasons represent the third most frequent category, affecting 133 PRs (22%). The dominant pattern in this category is CI/test failure, observed in 99 PRs (17%), where automated builds or tests fail due to the submitted changes. At times, reviewers even explicitly point out these failures, for example: “*@copilot fix the merge conflicts; if you cannot fix these then close the PR*” [16]. Incorrect implementations (19; 3%) and incomplete implementations (15; 2%) comprise the remaining code-level failures. Reviewers often highlight technical inaccuracies or missing logic with comments such as: “*The changes made to the billing.test.ts file are entirely wrong*” [17].

AGENTIC-level issues are the least frequent category, comprising 13 PRs (2%). Misalignment is the dominant pattern at this level, appearing in 9 PRs (1%). In these cases, agents repeatedly fail to

Table 2: Taxonomy of Rejection Patterns in Agentic PRs.

Level	Pattern	Definition	Freq.
REVIEWER	Abandoned/Not Reviewed	PR closed with no meaningful human interaction; only bots (if any) performed actions.	228
	Duplicate PR	Work already exists in another PR; maintainers explicitly reference the duplicate.	142
	Unwanted Feature	Maintainers state that the feature is unnecessary, misaligned, or not intended.	24
PULL REQUEST	Non-Functional PR	PR contains only setup, configuration, or scaffolding changes without functional contribution.	13
	Wrong Task Description	PR description reflects misunderstanding of the task or intended change.	7
	Wrong Branch	PR targets the incorrect branch and must be resubmitted to the proper one.	2
CODE	CI/Test Failure	PR fails automated build or deployment checks caused by its own changes.	99
	Incorrect Implementation	Implementation is technically incorrect or solves the wrong problem despite a correct task description.	19
	Incomplete Implementation	Contribution lacks required logic or completeness; reviewers flag missing or insufficient work.	15
AGENTIC	Misalignment	Agent fails to follow reviewer instructions or misunderstands explicit requested edits.	9
	License Issues	Reviewers flag license or ownership concerns related to agent-generated content.	4

follow explicit reviewer instructions or misunderstand requested changes, even after multiple rounds of feedback. Reviewer comments often express frustration with comments such as “*Devin stop being a dumb*ss, if you claim you “deleted 200 lines” then continue to*” [18]. Licensing issues account for the remaining 4 PRs (<1%). These PRs are rejected because agents do not comply with project-specific legal requirements, such as signing a Contributor License Agreement (CLA) or addressing ownership concerns. Maintainers explicitly reference these requirements with comments, e.g., “*we ask that you sign our Contributor License Agreement before we can accept your contribution*” [19]. Together, these examples highlight legal and governance constraints that current agents cannot satisfy.

A majority of agentic PRs are not merged due to reviewer abandonment. Among reviewed PRs, the dominant rejection patterns include duplicate PRs, CI/test failures, and large or unwanted feature implementations.

4 Conclusion

Our findings indicate that not-merged PRs tend to introduce larger and more invasive code changes, attempt broader feature additions, and exhibit higher rates of CI/test failures. Rejections of agentic PRs stem from multiple reasons, such as reviewer abandonment, duplicate PRs, or implementations of unwanted features. Overall, these results highlight difficulties of agents in task selection, coordination, and alignment with repository context. Maintainer feedback frequently emphasizes that PRs should be small, focused, and limited to a single coherent change, and discourages agentic submissions that combine substantive modifications with unrelated edits. The replication package for our study is publicly available [32].

Improving the success of future agentic-AI workflows would require improving agents’ ability to identify existing or ongoing work, adhere to project contribution norms, decompose tasks into localized changes, and validate submissions against CI pipelines before opening new PRs. Failures of agentic PRs can also be socio-technical rather than purely technical. By characterizing the failure patterns of not-merged agentics PRs, our study provides empirical grounding for the design of more context-aware and collaboration-sensitive AI coding agents, and informs future research on integrating such agents into real-world software development workflows.

References

- [1] Theophilus Azungah. 2018. Qualitative research: deductive and inductive approaches to data analysis. *Qualitative Research Journal* 18, 4 (2018), 383–400. doi:10.1108/QRJ-D-18-00035
- [2] Shraddha Barke, Michael B James, and Nadia Polikarpova. 2023. Grounded copilot: How programmers interact with code-generating models. *Proceedings of the ACM on Programming Languages* 7, OOPSLA1 (2023), 85–111.
- [3] Ira Ceka, Saurabh Pujar, Shyam Ramji, Luca Buratti, Gail Kaiser, and Baishakhi Ray. 2025. Understanding Software Engineering Agents Through the Lens of Traceability: An Empirical Study. arXiv:2506.08311 [cs.SE] <https://arxiv.org/abs/2506.08311>
- [4] Mark Chen. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021).
- [5] Mark Chen and Jerry Tworek et al. 2021. Evaluating Large Language Models Trained on Code. (2021). arXiv:2107.03374 [cs.LG]
- [6] William G. Cochran. 1977. *Sampling Techniques* (3rd ed.). John Wiley & Sons, New York, NY.
- [7] ConventionalCommits. 2025. <https://www.conventionalcommits.org/>
- [8] Ramtin Ehsani, Esteban Parra, Sonia Haiduc, and Preetha Chatterjee. 2025. Hierarchical Knowledge Injection for Improving LLM-based Program Repair. In *40th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. <https://arxiv.org/abs/2506.24015>
- [9] Ramtin Ehsani, Sakshi Pathak, Esteban Parra, Sonia Haiduc, and Preetha Chatterjee. 2025. What characteristics make ChatGPT effective for software issue resolution? An empirical study of task, project, and conversational signals in GitHub issues. *Empirical Software Engineering* 31, 1 (Nov. 2025). doi:10.1007/s10664-025-10745-8
- [10] GitHub. 2025. <https://github.com/graphistry/pygraphistry/pull/706>
- [11] GitHub. 2025. <https://github.com/rvunet/rv-FANN/pull/59>
- [12] GitHub. 2025. <https://github.com/bmad-code-org/BMAD-METHOD/pull/196>
- [13] GitHub. 2025. <https://github.com/coder/coder/pull/16917>
- [14] GitHub. 2025. <https://github.com/microsoft/vscode-cpptools/pull/13763>
- [15] GitHub. 2025. <https://github.com/getsetn/sentry-javascript/pull/16526>
- [16] GitHub. 2025. <https://github.com/hyperlight-dev/hyperlight/pull/641>
- [17] GitHub. 2025. <https://github.com/firecrawl/firecrawl/pull/1645>
- [18] GitHub. 2025. <https://github.com/reflex-dev/reflex-web/pull/1479>
- [19] GitHub. 2025. <https://github.com/netdata/netdata/pull/20631>
- [20] Marko A. Hofmann. 2015. Searching for effects in big data: Why p-values are not advised and what to use instead. In *2015 Winter Simulation Conference (WSC)*. 725–736. doi:10.1109/WSC.2015.7408210
- [21] Kosei Horikawa, Hao Li, Yutaro Kashiba, Bram Adams, Hajimu Iida, and Ahmed E. Hassan. 2025. Agentic Refactoring: An Empirical Study of AI Coding Agents. arXiv:2511.04824 [cs.SE] <https://arxiv.org/abs/2511.04824>
- [22] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. SWE-bench: Can Language Models Resolve Real-World GitHub Issues? arXiv:2310.06770 [cs.CL] <https://arxiv.org/abs/2310.06770>
- [23] Sungmin Kang, Gabin An, and Shin Yoo. 2024. A Quantitative and Qualitative Evaluation of LLM-Based Explainable Fault Localization. *Proc. ACM Softw. Eng.* 1, FSE, Article 64 (July 2024), 23 pages. doi:10.1145/3660771
- [24] Barbara Kitchenham, Lech Madeyski, David Budgen, Jacky Keung, Pearl Brereton, Stuart Charters, Shirley Gibbs, and Amnart Poonthong. 2017. Robust Statistical Methods for Empirical Software Engineering. *Empirical Software Engineering* 22, 2 (April 2017), 579–630. doi:10.1007/s10664-016-9437-5
- [25] Valentina Lenarduzzi, Vili Nikkola, Nyyti Saarimäki, and Davide Taibi. 2021. Does code quality affect pull request acceptance? An empirical study. *Journal of Systems and Software* 171 (2021), 110806. doi:10.1016/j.jss.2020.110806
- [26] Hao Li, Haoliang 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. arXiv:2507.15003 [cs.SE] <https://arxiv.org/abs/2507.15003>
- 15003
- [27] Rungroj Maipradit, Dong Wang, Patanamon Thongtanunam, Raula Gaikovina Kula, Yasutaka Kamei, and Shane McIntosh. 2023. Repeated Builds During Code Review: An Empirical Study of the OpenStack Community. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 153–165. doi:10.1109/ASE56229.2023.00030
- [28] Oorja Majgaonkar, Zhiwei Fei, Xiang Li, Federica Sarro, and He Ye. 2025. Understanding Code Agent Behaviour: An Empirical Study of Success and Failure Trajectories. arXiv:2511.00197 [cs.SE] <https://arxiv.org/abs/2511.00197>
- [29] Mary McHugh. 2012. Interrater reliability: The kappa statistic. *Biochemia medica : časopis Hrvatskoga društva medicinskih biokemičara / HDMB* 22 (10 2012), 276–82.
- [30] Noor Nashid, Daniel Ding, Keheliya Gallaba, Ahmed E. Hassan, and Ali Mesbah. 2025. Characterizing Multi-Hunk Patches: Divergence, Proximity, and LLM Repair Challenges. arXiv:2506.04418 [cs.SE] <https://arxiv.org/abs/2506.04418>
- [31] John Pangas, Suhaib Mujahid, Ahmad Abdellatif, and Marco Castelluccio. 2025. Using LLMs to Bridge the Gaps in QA Test Plans at Firefox. *IEEE Software* (2025), 1–7. doi:10.1109/MS.2025.3621128
- [32] ReplicationPackage. 2025. https://anonymous.4open.science/r/MSR2026_AIDev-035B/
- [33] Amirali Sajadi, Binh Le, Anh Nguyen, Kostadin Damevski, and Preetha Chatterjee. 2025. Do LLMs consider security? an empirical study on responses to programming questions. *Empirical Software Engineering* 30, 3 (2025), 101.
- [34] Noah Shimm, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems* 36 (2023), 8634–8652.
- [35] Daricelio Moreira Soares, Manoel L De Lima Junior, Leonardo Murta, and Alexandre Plastino. 2015. Rejection factors of pull requests filed by core team developers in software projects with high acceptance rates. In *2015 IEEE 14th international conference on machine learning and applications (ICMLA)*. IEEE, 960–965.
- [36] Gail M. Sullivan and Richard Feinn. 2012. Using Effect Size—or Why the P Value Is Not Enough. *Journal of Graduate Medical Education* 4, 3 (Sept. 2012), 279–282. doi:10.4300/JGME-D-12-00156.1
- [37] Michael C. Thrun, Tino Gehlert, and Alfred Ultsch. 2020. Analyzing the fine structure of distributions. *PLOS ONE* 15, 10 (Oct. 2020), e0238835. doi:10.1371/journal.pone.0238835
- [38] Erik van der Veen, Georgios Gousios, and Andy Zaidman. 2015. Automatically Prioritizing Pull Requests. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. 357–361. doi:10.1109/MSR.2015.40
- [39] Aidan Z. H. Yang, Claire Le Goues, Ruben Martins, and Vincent Hellendoorn. 2024. Large Language Models for Test-Free Fault Localization. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering (Lisbon, Portugal) (ICSE '24)*. Association for Computing Machinery, New York, NY, USA, Article 17, 12 pages. doi:10.1145/3597503.3623342
- [40] John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems* 37 (2024), 50528–50652.
- [41] Fiorella Zampetti, Gabriele Bavota, Gerardo Canfora, and Massimiliano Di Penta. 2019. A Study on the Interplay between Pull Request Review and Continuous Integration Builds. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 38–48. doi:10.1109/SANER.2019.8667996
- [42] Xunhui Zhang, Yue Yu, Georgios Gousios, and Ayushi Rastogi. 2023. Pull Request Decisions Explained: An Empirical Overview. *IEEE Transactions on Software Engineering* 49, 2 (2023), 849–871. doi:10.1109/TSE.2022.3165056
- [43] Yangyang Zhao, Alexander Serebrenik, Yuming Zhou, Vladimir Filkov, and Bogdan Vasilescu. 2017. The impact of continuous integration on other software development practices: a large-scale empirical study. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 60–71.