Why do you fail me, Mr. Bot?

Amaan Ahmed
a477ahme@uwaterloo.ca
University of Waterloo
Waterloo, Ontario, Canada

Asim Waheed
a7waheed@uwaterloo.ca
University of Waterloo
Waterloo, Ontario, Canada

Youssef Souati
ysouati@uwaterloo.ca
University of Waterloo
Waterloo, Ontario, Canada

Abstract

Autonomous coding agents now generate a large share of pull requests (PRs) on GitHub, yet it remains unclear how reliably these contributions succeed in real-world development workflows. Using the AIDev dataset, we study over 33,000 **agent-authored PRs from high-activity repositories and compare their outcomes against human-authored PRs. Our analysis shows that agents fail more often than humans in curated, actively maintained projects, with failure rates diverging sharply across task types. Agents perform well on generative tasks such as documentation and testing, but struggle on precision-oriented work including chores, configuration changes, and style-related updates. We further classify 7,646 failed agentic PRs and find that most failures stem from deprioritization or ambiguous outcomes rather than explicit technical rejection. When failures are technical, they are dominated by CI and test breakages rather than reviewer-detected correctness issues. Finally, we identify practices that shape agent success: smaller and more focused PRs succeed more often, while large diffs and wide architectural footprints correlate with failure; and agent contributions fare worse in highly popular repositories with stricter review processes. Taken together, our findings show that agent reliability depends less on model capability and more on project context, review bandwidth, and PR design. Improving the environment around agents through task scoping, CI feedback, and repository-specific adaptation may matter more than improving the agent itself.**

Keywords

Agentic Pull Requests; Large Language Models; Software Engineering; Empirical Study

1 Introduction

AI has become deeply embedded in modern software engineering. Developers now routinely rely on AI tools for tasks ranging from general problem solving [2] and code completion to even delegating entire issues to autonomous coding agents [3, 4]. As these agents take on increasingly substantive roles in real-world workflows, questions about their reliability and effectiveness become more pressing.

Recent evidence shows that autonomous agents such as Codex, Devin, Copilot, Cursor, and Claude Code now create hundreds of thousands of pull requests (PRs) across the open-source ecosystem [4]. Yet a striking proportion of these contributions fail to be merged, stall indefinitely, or are reverted shortly after landing. These patterns suggest that, despite their growing activity, current AI teammates still struggle to produce contributions that consistently meet project expectations and governance norms. Understanding *why* these failures occur—and the conditions under which agent-authored PRs do succeed—is essential for building trustworthy AI-assisted development workflows.

In this work, we analyze failure outcomes in agent-authored PRs using the AIDev dataset [4], which includes over 900,000 agentic PRs across more than 100,000 repositories. We further rely on a curated subset enriched with reviews, timelines, diffs, and issue links to support deeper analyses. Using this dataset, we investigate the following research questions:

(1) Are agent-authored PRs more or less likely to succeed than human-authored PRs?
(2) What factors lead to the rejection of agent-authored PRs?
(3) How can we identify practices that maximize the success of agent-authored PRs?

Our goal is to provide an empirical foundation for understanding how AI teammates behave in practice. We examine the failure modes they most commonly encounter. We investigate the repository contexts and PR characteristics that shape their outcomes.

2 Background

The reasons behind pull request (PR) rejection have long been studied in software engineering. Many rejections stem not from technical flaws but from process- or context-related issues. For example, redundancies are common in fork-based development [5], and many PRs are rejected because they duplicate existing changes [6]. Other studies note that rejection factors vary with project size, activity, and community dynamics [1]. Overall, rejection often depends more on *context*, such as timing, coordination, or relevance, than on code quality itself.

Zhang et al. find that the single strongest predictor of PR acceptance is whether the author is a core team member [7]. This highlights the social dimension of PR evaluation and the importance of trust, reputation, and reviewer familiarity. While these insights explain human-authored PRs, it remains unclear whether the same dynamics hold for agentic PRs, where authorship, identity, and accountability differ fundamentally.

Recent work shows that autonomous coding agents now create PRs at scale [4], raising new questions about how these contributions are perceived and evaluated by human maintainers. Because agents lack long-term participation histories and social standing, traditional predictors of acceptance may not apply. This gap motivates a deeper investigation into how and why agent-authored PRs fail in real-world development workflows.

CS 846, Waterloo, ON, Canada
2025. ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

3 Methodology

Dataset Details. AIDev provides two datasets of agent-authored pull requests. The **full dataset** contains 932,791 PRs across 116,211 repositories and is used only for high-level statistics. The **curated dataset** contains 33,596 PRs from 2,807 repositories with at least 100 stars and includes richer artifacts, such as reviews, inline comments, commit-level diffs, issues, and PR timelines. Task-type annotations are available for all PRs and are used throughout our study.

Defining Failed PRs. A PR is considered *failed* if it is closed without being merged or remains open but inactive for more than 180 days. This yields 7,270 closed-but-unmerged PRs and 376 stale ones, for a total of 7,646 failed agentic PRs.

3.1 RQ1: Comparing Human-Authored PRs.

We extract 6,618 human-authored PRs from the curated dataset, spanning 818 repositories and 2,515 users. Applying the same failure definition yields 1,204 failed human PRs. Human PRs include only PR-level metadata and task-type annotations.

3.2 RQ2: Identifying Reasons for Rejection.

Failure taxonomy. Recall that the goal of RQ2 is to identify the factors that lead to the rejection of agent-authored pull requests. To do this, we first need to understand the possible reasons why a PR may fail. Prior work provides a useful starting point for constructing such a taxonomy [6].

Because our dataset provides less contextual detail than Gerrit, on which prior work is based on, we refine the categories to ensure they can be applied consistently to GitHub PRs. This process yields the taxonomy shown in Table 1, which groups failure reasons into technical issues, fit or value concerns, process or ownership factors, redundancy or obsolescence, invalid submissions, and cases where the reason cannot be determined. We encode each category as a structured JSON definition used directly in our LLM prompt, specifying the meaning of each label and the signals the model should look for (Figure 1).

Gathering context. To enable the LLM to classify the reason for failure, we collect all contextual information available for each PR in the curated dataset. The information we gather includes the following:

- **Repository metadata**: project name, popularity indicators (e.g., stars), primary language, and other attributes that situate the PR within its development environment.
- **Author metadata**: limited information about the PR author, including account age and activity, as exposed by the dataset.
- **Diff summary**: a high-level summary of the files changed, additions, deletions, and structural modifications made by the PR.
- **Timeline events**: the sequence of actions on the PR (e.g., opened, reviewed, CI triggered, labeled, closed), providing temporal and procedural context.
- **CI / workflow signals**: evidence of failed or successful automated checks, build failures, and test results when available.
- **Review text**: comments from reviewers, review-thread discussions, inline review comments, and general PR discussion; we deduplicate repeated text such as bot messages.

```
{
  "id": "T1_CI_OR_TEST_FAILURE",
  "group": "TECHNICAL",
  "definition": "The PR is not merged mainly
      because tests, builds, or CI checks fail due
       to issues introduced or exposed by the PR,
      and the author never fixes them.",
  "typical_signals": [
    "ci_failed = true",
    "Timeline events like 'ci_failure', '
        workflow_run.completed: failure'",
    "Review comments: 'tests are failing', 'please
         fix CI', 'build is broken'",
    "No later commit that fixes the failures
        before close"
  ]
}
```

Figure 1: Example JSON definition used to guide the LLM classifier for category T1.

- **Commit messages**: short natural-language summaries of individual commits associated with the PR.
- **PR outcome**: the final status of the PR (merged, closed, or left open), which determines whether the PR enters our failure analysis.

Prompting strategy. For every pull request, we construct a single text file that includes the PR title and description, a diff summary, the event timeline, CI and workflow signals, review and discussion text, commit messages, and the final outcome. This unified representation ensures that the model has access to all information that a human reviewer would typically consult when assessing why a PR did not merge.

The LLM operates using a taxonomy-guided classification prompt. We supply the full taxonomy of failure reasons, along with concise natural-language definitions of each category and the behavioural signals associated with them. To enforce consistency and discourage hallucination, the prompt specifies strict output constraints and instructs the model to select exactly one category for each PR. When insufficient evidence is present, the model is explicitly required to use the `U1_UNKNOWN` label rather than guessing.

The prompt also includes an explicit JSON schema that the model must follow. The schema specifies four fields: the PR identifier, the selected reason label, a short natural-language justification, and a numerical confidence score between 0 and 1. By constraining the output format and limiting the task to single-label classification, we reduce the likelihood of malformed or ambiguous model responses.

This approach is best understood as a form of one-shot learning. The prompt provides a single worked example of how categories are defined and how the model should interpret contextual signals, after which the model applies these rules to new PRs. Each PR is therefore evaluated independently using the same structured prompt and its corresponding context file. In our experiments, we use both `Llama-3.2-3b` and `Qwen-2.5` as the underlying classification models.

Group	Description	Category
TECHNICAL (T)	Code, tests, style, architecture	T1: CI or Test Failure T2: Code Quality or Correctness
FIT_OR_VALUE (F)	Feature/requirement mismatch, not needed, wrong direction	F1: Wrong feature, not needed, or misaligned
PROCESS_OR_OWNERSHIP (P)	Time, priority, unresponsiveness, social/review issues, policy	P1: Low priority P2: Author Unresponsive P3: Review/social/coordination conflict P4: Policy/compliance/bureaucracy
REDUNDANCY_OR_OBSOLETE (R)	Duplicate feature or superseded work	R1: Duplicate feature R2: Obsolete PR
INVALID_OR_SPAM (S)	Spam, noise, non-meaningful PRs	S1: Invalid/Spam/Nonsensical
UNKNOWN (U)	Not enough information	U1: Unknown

Table 1: Failure taxonomy used for classifying rejected PRs.

3.3 RQ3

To answer RQ3, we use the merged dataset from RQ2 and analyze the commit-level, review-level, timeline-level, and repository-level metadata. We join failure labels and task types with structural features from `pr_commit_details`, review dynamics from `pr_reviews`, lifecycle information from `pr_timeline`, and repository attributes such as stars, forks, and language. These combined signals allow us to study how task type, PR size, architectural footprint, review activity, and project characteristics relate to failure reasons and overall success.

4 Results

4.1 RQ1

Here we answer the question: **Are agent-authored PRs more or less likely to succeed than human-authored PRs?**

Overall failure-rate comparison. Across the curated subset of high-activity repositories, agent-authored pull requests fail more often than human-authored ones. Agents fail in **22.79%** of their PRs. Humans fail in **19.70%** of their PRs. In the full AIDev dataset, agents show a much lower failure rate of **8.18%**, but this is driven by dataset composition rather than true performance. The full dataset contains many small or low-governance repositories, where PRs often merge with minimal scrutiny and where agents are overrepresented. The curated subset, by contrast, contains popular, actively maintained repositories with established review processes and CI. For all remaining analyses of RQ1, we therefore rely only on the curated subset.

Breakdown by task type. We next analyze whether agents and humans differ in how they are used across development tasks (Figure 2a). Agents primarily contribute to **features** (43.0%), **bug fixes** (24.1%), **documentation** (11.6%), **testing** (7.0%), and **refactoring** (6.8%). Humans, by contrast, contribute a larger share of **build system changes** (9.3% vs. 1.9%) and **chore and maintenance tasks** (12.8% vs. 2.7%), as well as other infrastructure work. These differences suggest that agents are used primarily for generative or patch-style tasks, whereas humans handle more infrastructure-oriented and context-dependent work.

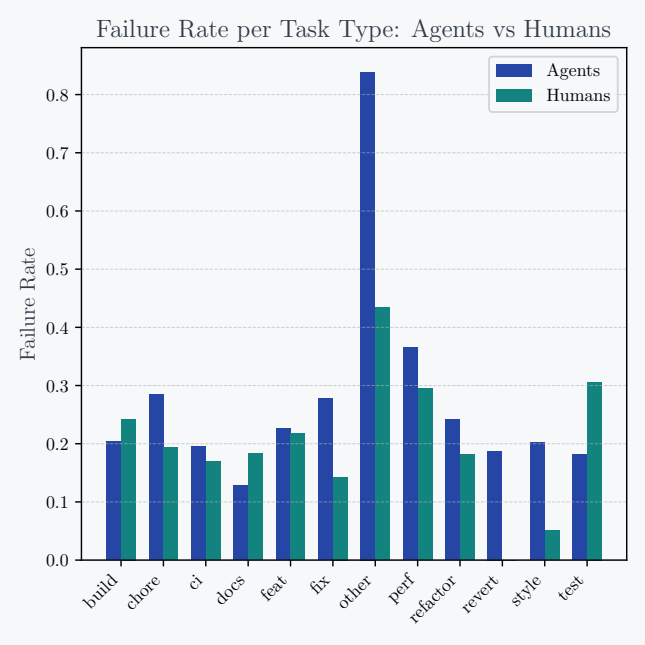
Failure-rate differences by task type. Failure rates vary substantially across task categories, highlighting the strengths and weaknesses of agents compared to humans (Figure 2b). Agents perform worse than humans on maintenance tasks. Agent-authored **chore** PRs fail at **28.46%**, compared to **19.36%** for humans, and **performance** patches fail at **36.47%** vs. **29.55%**. Agents also fail far more often in the **other** category (**83.87%** vs. **43.48%**), which captures ambiguous or irregular PR types. These patterns suggest that agents struggle when tasks require nuanced contextual understanding or cross-cutting architectural reasoning.

Humans, on the other hand, perform worse on several generative or boilerplate-heavy tasks. Human-authored **tests** fail at **30.60%**, compared to **18.16%** for agents, and **documentation** tasks fail at **18.30%** vs. **12.76%** for agents. Humans also fail more often in **build** changes (**24.27%** vs. **20.41%**). These tasks tend to involve repetitive structure, consistent formatting, and strict adherence to conventions, which agents handle well.

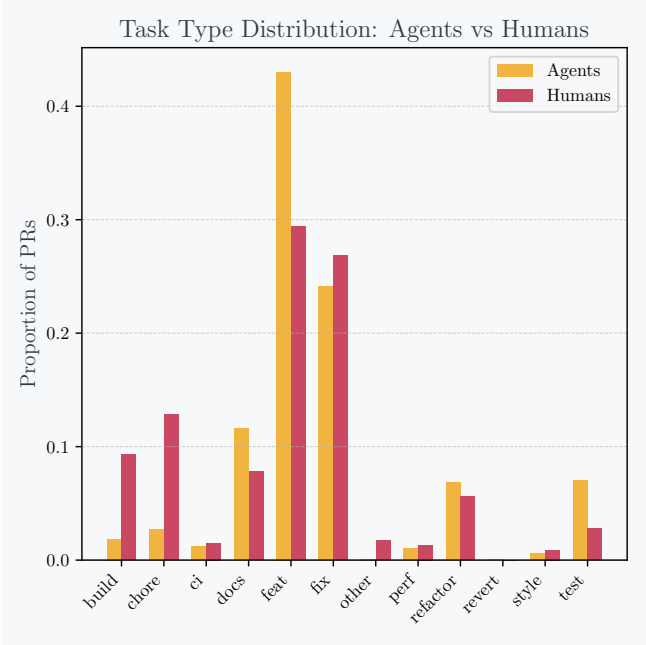
One category shows a particularly sharp divide: **style**-related PRs. Agents fail style checks at **20.21%**, whereas humans fail at only **5.08%**. This gap reflects that humans internalize project-specific style conventions, while agents often generate syntactically valid code that nonetheless violates linting or formatting requirements.

Summary of insights. Our findings reveal clear and interpretable patterns:

- **Insight 1: Agents are predominantly used for generative tasks.** They submit a much higher proportion of feature, documentation, testing, and refactoring PRs, which align well with LLM strengths.
- **Insight 2: Agents struggle on precision-oriented maintenance work.** Tasks such as chores, CI configuration, and performance tuning require nuanced repository context and exactness, and these categories exhibit higher failure rates for agents.



(a) Task-type distribution for agent- and human-authored PRs.



(b) Failure rates by task type for agents and humans.

Figure 2: Comparison of PR task-type distributions and failure-rate patterns between agent- and human-authored pull requests.

- **Insight 3: Humans struggle on tedious boilerplate-heavy tasks.** Humans fail more often on documentation, test creation, and build configuration, tasks that benefit from the consistency and structure that agents provide.
- **Insight 4: Style is the clearest point of divergence.** Humans almost never fail style checks, whereas agents fail **four times more often**, likely due to mismatches with project-specific linting rules.
- **Insight 5: Agents fail disproportionately in ambiguous or atypical categories.** Very high failure rates in categories such as *other* and *revert* indicate difficulty handling tasks that require contextual judgment or intent reconstruction.

Together, these results show that agents and humans occupy different roles in modern development workflows, and that their strengths and weaknesses are strongly task-dependent.

4.2 RQ2

Here we answer the question: **What factors lead to the rejection of agent-authored PRs?**

Model agreement and divergence. We first classify reasons for failure using two large language models, LLaMA and Qwen, and compare how they label a shared set of 7,646 failed PRs. Their agreement is only **44.7%**, indicating substantial divergence in how each model interprets the underlying cause of failure. The confusion matrix in Figure 3 shows that both models, but especially Qwen, frequently map diverse failure types into the **P1** category ("Low Priority" or "Left inactive until stale"), while LLaMA spreads these same PRs across more specific technical, process, redundancy, and fit- or value-related categories. Disagreement is therefore structured

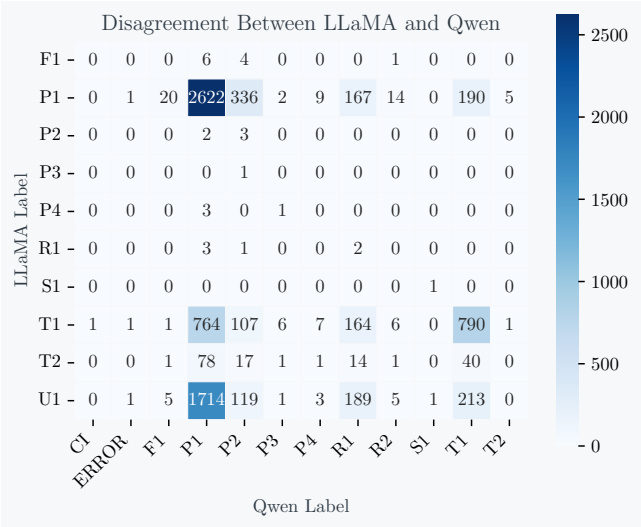


Figure 3: Confusion matrix of failure-reason labels assigned by LLaMA (rows) and Qwen (columns) for failed pull requests.

rather than random, and is dominated by asymmetric flows into and out of *P1*.

Combining model outputs. Because disagreement is systematic, simply choosing one model as ground truth would bias the analysis. Using only Qwen would collapse many failures into *P1*, while using only LLaMA would emphasize fine-grained distinctions that Qwen

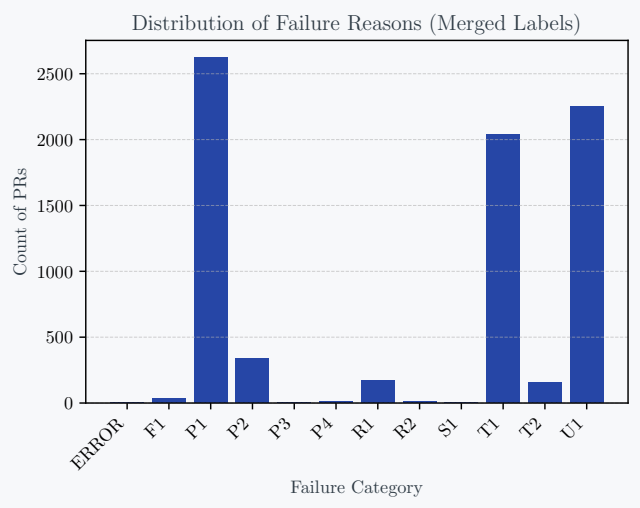


Figure 4: Distribution of merged failure-reason labels for failed pull requests.

often ignores. We therefore treat **P1** as a "weak" label. When one model assigns a specific category and the other assigns *P1*, we keep the specific label. Only when **both** models assign *P1* do we treat the PR as genuinely low priority. This reconciliation preserves information where either model provides it and yields a more balanced view of failure reasons for downstream analysis.

Distribution of failure reasons. After merging labels, we examine the distribution of failure categories across all failed PRs (Figure 4). The distribution is highly skewed. **P1 (Low Priority)** is the most common reason, with **2,622** PRs where changes were left inactive, deprioritized, or effectively abandoned. The next most common category is **U1 (Unknown)** with **2,251** PRs, where neither model could infer a clear rationale, often due to sparse metadata or limited discussion.

Technical failures are also prominent. **T1 (CI or Test Failure)** appears in **2,038** PRs, indicating that many failed contributions never pass automated checks. By contrast, **T2 (Code Quality or Correctness)** appears in only **158** cases, suggesting that explicit reviewer-detected logic errors or style issues are relatively rare compared to CI-related failures.

Process- or ownership-related issues beyond low priority are uncommon. **P2 (Author Unresponsive)** appears in **341** cases, while **P3** (coordination or conflict) and **P4** (policy or compliance) appear in only **3** and **13** cases respectively. Redundancy plays a limited role, with **R1 (Duplicate)** in **173** cases and **R2 (Obsolete PR)** in **14**. Explicitly invalid or spam submissions (**S1**) are extremely rare, with only one detected instance.

Overall, three patterns emerge. First, many PRs fail due to simple deprioritization rather than explicit technical rejection. Second, technical failures are dominated by CI and test issues rather than reviewer-detected logic errors. Third, a substantial fraction of PRs fail for reasons that remain ambiguous or poorly documented. These reasons are in line with why human-authored PRs fail based on prior work [].

Summary of insights. Our analysis of failure reasons yields several key insights:

- **Insight 1: LLM-based labels disagree in structured ways.** Qwen tends to collapse diverse failures into *P1*, whereas LLaMA preserves finer distinctions among technical, process, redundancy, and fit-related causes.
- **Insight 2: Treating low-priority labels as "weak" recovers otherwise lost information.** Our reconciliation strategy leverages specific labels whenever either model provides them, avoiding both over-collapsing and over-fragmentation.
- **Insight 3: Most failures reflect deprioritization rather than explicit rejection.** *P1* dominates the distribution, indicating that many PRs simply age out due to limited attention or perceived low impact.
- **Insight 4: CI and tests are the main technical gatekeepers.** *T1* is far more common than *T2*, showing that automated CI failures are a more frequent cause of rejection than reviewer-detected logic or style issues.
- **Insight 5: Social and policy conflicts are rare.** Categories such as *P3* and *P4* occur infrequently, suggesting that most failures are driven by prioritization and technical factors rather than interpersonal or governance disputes.

These results provide the failure taxonomy that RQ3 builds on when examining which task types, PR characteristics, and repository contexts are most conducive to agent success.

4.3 RQ3

Here we answer the question: **Can we identify practices that maximize the success of agent-authored PRs?**

How does the type of task influence the reason for failure? We first examine how failure reasons differ by task type for agent-authored PRs. Across most categories, the dominant reasons are **P1 (Low Priority)** and **U1 (Unknown)**, indicating that many agent submissions are closed without a lot of engagement rather than due to clear technical faults. This is especially true for *docs*, *chore*, *refactor*, and *build* tasks, which appear easy for agents to generate but relatively low-impact from the maintainers' perspective.

For tasks that modify executable code, such as *feat* and *fix*, failures are instead dominated by **T1 (CI or Test Failure)** and, to a lesser extent, **T2 (Code Quality or Correctness)**. These categories involve hundreds of CI-related failures, suggesting that agents frequently introduce behavior that breaks tests, violates implicit constraints, or triggers environment-specific issues. Refactoring exhibits a similar pattern, highlighting that even mechanical code transformations often have unintended consequences.

Overall, failures cluster around two broad mechanisms. Low-impact tasks are often deprioritized and closed as *P1* or *U1*, while code-modifying tasks are more likely to fail due to CI or correctness issues. These patterns suggest that improving agent success will require task-specific practices, such as providing stronger justification for low-impact PRs and adding automated validation or self-review loops for changes that affect code behavior.

Affect of PR size. A clear pattern emerges when comparing the size of successful and failed agent-authored pull requests. Successful PRs are much smaller, with a median of **168** added lines and

23 deleted lines, compared to **350** additions and **40** deletions for failed PRs. Failed PRs also touch more files (median of **4** versus **3**), indicating a wider architectural footprint. Although both groups typically contain only a single commit, the overall diff size and file spread differ sharply. These results suggest that agent PRs succeed when they are narrow in scope and easy for maintainers to review, while large, multi-file changes—especially in a single commit—are far more likely to be ignored, rejected, or to trigger CI failures.

Repository characteristics. Repository popularity also correlates with the success of agent-authored pull requests. Successful PRs come from repositories with a median of **306** stars and **58** forks, whereas failed PRs come from substantially larger projects with medians of **750.5** stars and **165** forks. This pattern indicates that agents perform better in smaller or less mature repositories and struggle in large, high-visibility projects with stricter governance and heavier review activity. Agent contributions therefore appear most effective in lower-governance settings, while deployments in popular projects may require additional support, oversight, or tooling.

Summary of insights. Taken together, our RQ3 analyses highlight concrete practices and contexts that shape agent success:

- **Insight 1: Low-impact tasks often fail silently.** For *docs*, *chore*, *refactor*, and *build* PRs, failures are dominated by *P1* and *U1*, indicating closure without substantial engagement rather than explicit technical rejection.
- **Insight 2: Code-changing tasks fail primarily at the CI gate.** For *feat* and *fix* PRs, *T1* and *T2* are the main failure reasons, showing that agent changes frequently break tests or violate implicit correctness constraints.
- **Insight 3: Smaller, focused PRs are more likely to succeed.** Successful agent PRs have fewer added and deleted lines and touch fewer files, suggesting that constrained scope and limited footprint make agent contributions easier to review and accept.
- **Insight 4: Highly popular repositories are harder environments for agents.** Agents are more successful in repositories with fewer stars and forks, and they struggle in large, high-visibility projects with stricter governance and more competition for maintainer attention.
- **Insight 5: Task- and context-aware practices are necessary for reliable deployment.** Improving agent success will require tailoring workflows to task type and repository context, including stronger justification for low-impact changes and more robust validation for behavior-changing PRs.

Together with RQ1 and RQ2, these findings show that agent success is not only model-dependent but also shaped by what the agent is asked to do, how large and intrusive its changes are, and where in the ecosystem those changes are proposed.

5 Threats to Validity

Human–agent dataset imbalance. The curated dataset contains far fewer human-authored PRs than agent-authored ones, which limits the statistical power of direct comparisons in RQ1. Although we report normalized rates, the smaller human baseline may obscure subtler behavioral differences.

LLM labeling disagreement. Our failure-reason labels depend on LLaMA and Qwen, which show substantial disagreement due to limited contextual information for many PRs. While our reconciliation strategy mitigates this issue, some labels may still reflect model biases rather than ground-truth maintainer intent.

LLM classification accuracy. We did not have a manual ground truth dataset to evaluate the LLM classification accuracy. While we manually confirmed a few results ($\approx$ 10), we do not have a quantitative assessment on how well the LLMs classified the failure reason.

Differences between full and curated datasets. The full AIDev dataset includes many low-governance repositories, whereas the curated dataset focuses on popular, actively maintained projects. Findings derived from the curated subset may therefore not generalize to the broader ecosystem represented in the full dataset.

Limited PR context in the dataset. The AIDev dataset omits certain signals available in GitHub, such as full CI logs, inline diffs, and discussion threads truncated by bots or deleted comments. Missing context may influence both failure-labeling accuracy and downstream interpretations.

Potential misclassification of stale PRs. We treat PRs inactive for more than 180 days as failed, but some of these may have been informally accepted, deprioritized for legitimate project reasons, or left open intentionally. This heuristic may slightly overestimate failure rates.

External validity of agent behavior. Agent-authored PRs in the AIDev dataset come from a specific set of tools, prompts, and time periods. Their behavior may not fully represent future agent workflows, different prompting setups, or emerging LLM-based code agents.

6 Discussion

Stepping back from the results, the bigger story is surprisingly straightforward. Agent success has less to do with the agent itself and more to do with the project environment it lands in. Most failures are not dramatic technical issues but simple deprioritization or inattention. In that sense, agents experience the same open-source reality humans do.

Agents and humans struggle in different ways. Agents falter on tasks that need contextual judgment, while humans falter on tasks that need structure and patience. CI ends up being the main decision-maker rather than human reviewers. And when agents do succeed, it is usually because their PRs are small, focused, and submitted to repositories that are not overwhelmed.

The overall takeaway is that project management matters more than model intelligence, as mentioned in prior work [6]. Clear processes, manageable PR sizes, and healthy reviewer bandwidth help agents just as much as humans. Improving the environment around agents may be the most reliable path to improving their contributions.

References

[1] Tanay Gottigundala, Siriwan Sereesathien, and Bruno Silva. [n. d.]. Qualitatively Analyzing PR Rejection Reasons from Conversations in Open-Source Projects. In *2021 IEEE/ACM 13th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)* (2021-05). 109–112. https://doi.org/10.1109/CHASE52884.2021.00021

[2] Huizi Hao, Kazi Amit Hasan, Hong Qin, Marcos Macedo, Yuan Tian, Steven H. H. Ding, and Ahmed E. Hassan. [n. d.]. An Empirical Study on Developers' Shared Conversations with ChatGPT in GitHub Pull Requests and Issues. 29, 6 ([n. d.]), 150. https://doi.org/10.1007/s10664-024-10540-x

[3] Ahmed E. Hassan, Gustavo A. Oliva, Dayi Lin, Boyuan Chen, Zhen Ming, and Jiang. [n. d.]. *Towards AI-Native Software Engineering (SE 3.0): A Vision and a Challenge Roadmap.* https://doi.org/10.48550/arXiv.2410.06107 arXiv:2410.06107 [cs]

[4] Hao Li, Haoxiang Zhang, and Ahmed E. Hassan. [n. d.]. *The Rise of AI Teammates in Software Engineering (SE) 3.0: How Autonomous Coding Agents Are Reshaping Software Engineering.* https://doi.org/10.48550/arXiv.2507.15003 arXiv:2507.15003 [cs]

[5] Luyao Ren, Shurui Zhou, Christian Kästner, and Andrzej Wąsowski. [n. d.]. Identifying Redundancies in Fork-based Development. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)* (2019-02). 230–241. https://doi.org/10.1109/SANER.2019.8668023

[6] Qingye Wang, Xin Xia, David Lo, and Shanping Li. [n. d.]. Why Is My Code Change Abandoned? 110 ([n. d.]), 108–120. https://doi.org/10.1016/j.infsof.2019.02.007

[7] Xunhui Zhang, Yue Yu, Georgios Gousios, and Ayushi Rastogi. [n. d.]. Pull Request Decisions Explained: An Empirical Overview. 49, 2 ([n. d.]), 849–871. https://doi.org/10.1109/TSE.2022.3165056

A Additional JSON Definitions

ID	Group	Short definition
T1	TECHNICAL	CI or tests fail and are never fixed.
T2	TECHNICAL	Code-level problems: bugs, unsafe changes, or poor style.
F1	FIT_OR_VALUE	Change does not align with project goals or priorities.
P1	PROCESS	Low priority or left inactive until stale.
P2	PROCESS	Author stops responding or explicitly abandons the PR.
P3	PROCESS	Review or social disagreement, ownership confusion.
P4	PROCESS	Policy or compliance issues (CLA, wrong branch, missing issue).
R1	REDUNDANCY	Duplicate of another PR or already fixed.
R2	REDUNDANCY	Obsolete due to project or codebase changes.
S1	INVALID_OR_SPAM	Spam, nonsense, or mistaken PRs.
U1	UNKNOWN	Insufficient information to infer a reason.

Table 2: Compact summary of all failure categories and their intended meaning.

B Use of AI Tools

B.1 Creating Plots in Python

For the plots used in the paper, we used the help of ChatGPT. Specifically, we told ChatGPT the plot we want and the dataset we have, and it would generate the code required to create our plot.

B.2 Finding prior work

We used the help of ChatGPT to find prior work, such as the paper titled "Why is my code change abandoned?".

B.3 LaTeX-specific formatting

We used ChatGPT to help format the tables and figures used in the paper.

B.4 Debugging

While writing the code for our analyses, we used ChatGPT to quickly debug errors.