

On the Reliability of Agentic AI in Continuous Integration Pipelines

Moataz Chouchen

Concordia University
Montreal, Canada

moataz.chouchen@concordia.ca

Jasem Khelifi

ÉTS Montreal, University of Quebec
Montreal, Canada

jasem.khelifi.1@ens.etsmtl.ca

Mahi Begoug

ÉTS Montreal, University of Quebec
Montreal, Canada

mahi.begoug.1@ens.etsmtl.ca

Ali Ouni

ÉTS Montreal, University of Quebec
Montreal, Canada

Ali.Ouni@etsmtl.ca

Mohammed Sayagh

ÉTS Montreal, University of Quebec
Montreal, Canada

Mohammed.Sayagh@etsmtl.ca

Mohamed Aymen Saied

Laval university
Quebec city, Canada

mohamed-aymen.saied@ift.ulaval.ca

Abstract

Agentic AI systems powered by Large Language Models (LLMs) are increasingly used to autonomously contribute code in modern software development. While prior work has shown that such systems can accelerate development tasks, their reliability and maintenance behavior in real-world Continuous Integration (CI) workflows remain poorly understood. In this study, we analyze 11,771 pull requests (PRs) from GitHub, including 7,619 agentic and 4,152 human-authored PRs, to investigate how agentic code behaves during CI workflows. We examine (1) CI failure rates at the pull-request level, (2) responsibility for introducing and fixing CI failures, and (3) time-to-fix at the commit level using fail-fix mappings. Our results show that human-authored CI fixes exhibit a median time to fix of 71.70 minutes, whereas AI agentic-authored CI fixes resolve failures nearly four times faster, with a median of 17.23 minutes. Our results show that agent-authored fixes resolve CI failures nearly four times faster than human fixes (median 17.23 vs. 71.70 minutes). However, agents introduce most CI failures (79.15%) while performing a smaller share of fixes (60.63%), indicating that human developers remain heavily involved in failure resolution despite faster agent responses.

CCS Concepts

• **Software and its engineering** → **Software verification and validation**; *Continuous integration*; *Software maintenance tools*.

Keywords

Agentic AI, Continuous Integration, GitHub Actions

ACM Reference Format:

Moataz Chouchen, Jasem Khelifi, Mahi Begoug, Ali Ouni, Mohammed Sayagh, and Mohamed Aymen Saied. 2026. On the Reliability of Agentic AI in Continuous Integration Pipelines. In *23rd International Conference on Mining Software Repositories (MSR '26)*, April 13–14, 2026, Rio de Janeiro, Brazil. ACM, New York, NY, USA, 5 pages. <https://doi.org/10.1145/3793302.3793585>



This work is licensed under a Creative Commons Attribution 4.0 International License. *MSR '26, Rio de Janeiro, Brazil*

© 2026 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-2474-9/26/04

<https://doi.org/10.1145/3793302.3793585>

1 Introduction

In modern software engineering, DevOps practices emphasize rapid iteration, automation, and continuous delivery [5]. As a result, software changes are routinely validated through CI pipelines that automatically build, test, and validate code to provide fast feedback on integration failures [6]. Within this context, agentic AI systems such as Lovable, Gemini CLI, Claude-Code, Cursor, Base44, Codex, and Devin have shown the potential to substantially accelerate development workflows by autonomously performing tasks traditionally performed by human developers [14, 15]. These tasks include navigating and understanding codebases, generating and modifying source code, planning execution steps, running programs, and performing automated tests [10]. Despite the rapid adoption of agentic AI, an emerging body of research has raised concerns about the reliability and trustworthiness of LLM-based agents in software engineering tasks [13, 19].

For example, in a Copilot-generated PR in `microsoft/qdk` [16], the AI agent repeatedly attempted to resolve CI test failures, leading to at least 3 “try again” prompts from a developer. While the AI agent fixed several code and test-pattern issues, the failures ultimately stemmed from CI environment constraints related to blocked network access and insufficient timeouts, which were resolved only after a human adjusted the CI configuration.

Although research efforts investigated the use of AI agents to improve development workflows [9, 24], little is known about how agentic AI systems introduce CI failures in practice, nor about the time required for these systems to diagnose and repair such failures once they occur. This is critical to the development life cycle since failing CI requires developers a large amount of time and focus [12, 17, 20] and cause long delays in the feedback loop [18, 21].

To address this gap, we aim in this paper to investigate how reliable Agentic AI is in CI pipelines, shedding light on its failure frequency, the involvement of agentic AI and human developers in fixing these failures, and the time to fix them compared to human developers. To achieve this, we augment the PR data from the AIDev dataset [14] by collecting and analyzing 151,400 GitHub Actions workflow runs from agentic-authored PRs generated by Codex, Cursor, Copilot, Claude Code, and Devin, and 94,425 runs from human-authored PRs, analyzed at both the pull-request and commit levels. The following research questions guide this study:

- **RQ1.** *To what extent do builds fail in agentic pull requests compared to human pull requests?*
- **RQ2.** *Who introduces and repairs CI failures in agentic pull requests?*
- **RQ3.** *How does the time to fix CI failures differ between AI agent and human commits?*

Our results show that agentic PRs have slightly higher CI failure rates than human PRs at 27.23% versus 20.27%, with Copilot and Codex exhibiting failure rates comparable to humans. Although most CI failures in agentic PRs are introduced by agents at 79.15%, they perform a smaller share of the fixes at 60.63%, indicating human involvement in failure resolution. At the same time, agent-authored fixes are much faster, with a median time to fix of 17.23 minutes compared to 71.70 minutes for human PRs. These findings suggest that while agentic AI resolves CI failures quickly, human developers remain more involved in fixing CI failures than in introducing them.

Replication package: We provide a package of our dataset, scripts and experiments [1].

2 Study Design

Our goal is to *empirically* assess the reliability and repair behavior of agentic AI systems in CI workflows. To achieve this, our study covers three essential phases: (1) Data Collection, (2) CI workflow filtering, and (3) fail-fix pair mapping. In the following, we provide an overview of each phase.

2.1 Data Collection

We base our analysis on the AIDev dataset updated on Oct 16, 2025 [14]. Specifically, we use the AIDev-Pop subset, which contains 33,596 agentic PRs and 6,618 human PRs from popular projects. To ensure comparability between the two groups, we first applied a repository-level filter, keeping only PRs originating from projects with 500 or more GitHub stars. This filtering reduced agentic PRs from 33k to 12,373 PRs, matching the selection criteria already applied to the human PR subset [14]. For each remaining PR, we mapped all associated commits from the commit-details data, then used the GitHub REST API [8] to retrieve GitHub Actions workflow run metadata for each commit. The workflow metadata collected includes run outcomes, start and completion timestamps, workflow names, YAML file paths, and detailed job-level data.

2.2 CI workflows filtering

To ensure the analyzed workflow runs correspond to CI activities, we applied filtering criteria as described by Bernardo et al. [2]. A workflow run is considered CI-related if it satisfies these conditions:

- ▼ **Filter #1: Trigger event:** The workflow must be triggered by either a push or pull_request event, as CI workflows are typically executed in response to code changes introduced through commits or PRs.
- ▼ **Filter #2: Non-documentation workflow:** The workflow name or its YAML file path must not contain any documentation-related terms, including doc:, docs, documentation, or doc[to exclude runs that update documentation rather than source code.
- ▼ **Filter #3: CI-related keywords:** At least one of the following terms must appear in the workflow name, YAML file path,

job names, or step names: build, deploy, test, ci, continuous integration, or continuous_integration. For the keyword ci, we explicitly exclude cases where it appears only as part of another word (e.g., dependencies).

After applying these filters and retaining only PRs that execute at least one CI-related workflow run, our final dataset consists of 7,619 agentic PRs and 4,152 human PRs.

2.3 Mapping fail-fix pairs

To enable a fine-grained analysis of CI workflow failures and fixes in PR, we map CI failure and fix events to determine who introduces failures, who fixes them, and how long fixes take, which is required to answer RQ2 and RQ3. To avoid ambiguity in attributing commits to agents or humans at the commit level, we restrict our analysis to agentic PRs where committer identities can be reliably distinguished. We therefore exclude Codex PRs since it only attaches a "Codex" label to the PR and does not expose the AI agent committer identity at the commit level, resulting in 4,876 remaining agentic PRs [25].

✎ **Procedure of the mapping.** For the remaining agentic PRs, we construct fail-fix pairs following the approach of Tomassi et al. [23]. For each PR, workflow runs are grouped by workflow identifier and branch, then ordered chronologically. When a failed run is observed, we scan forward until the next successful run of the same workflow occurs. This failure and its subsequent success form a fail-fix pair. The process then continues from that success until no runs remain. This procedure initially yields 2,082 fail-fix pairs from agentic PRs and 1,599 from human PRs.

▼ **Ensuring correct agent-human attribution.** To reliably distinguish agentic and human contributions, we remove all fail-fix pairs in which either the failing or fixing commit is authored or committed by a GitHub account labeled [bot], as such accounts often correspond to CI infrastructure or third-party automation rather than autonomous coding agents. We make an exception for the officially recognized Devin identity (devin-ai-integration[bot]), which explicitly represents an agentic system. We retain only commits attributable to known agentic tools—Copilot, cursoragent, and devin-ai-integration[bot]. Claude Code commits are identified via the presence of a "co-authored-by: claude" marker in the commit message, enabling attribution despite the absence of a dedicated bot account. After cleaning, 365 bot-related fail-fix pairs are removed, yielding 1,717 agentic fail-fix pairs. Applying the same procedure to human-authored PRs yields 1,599 fail-fix pairs as a human baseline.

3 Study Results

3.1 RQ1. To what extent do builds fail in agentic pull requests compared to human pull requests?

Approach. Using the CI-filtered workflow runs described in Section 2.2, we compute for each PR its CI failure rate, defined as the proportion of failed CI workflow runs relative to all completed CI runs (success or failure). Only CI runs with passing or failing outcomes were considered, as canceled or timed-out runs frequently introduce environmental noise into build failure analyses [7, 11].

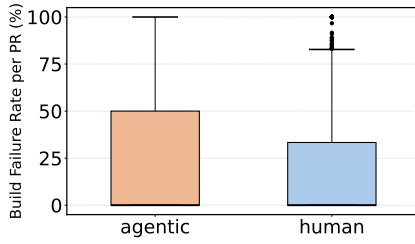


Figure 1: Distribution of build failure rates for human PRs vs. all agentic PRs combined.

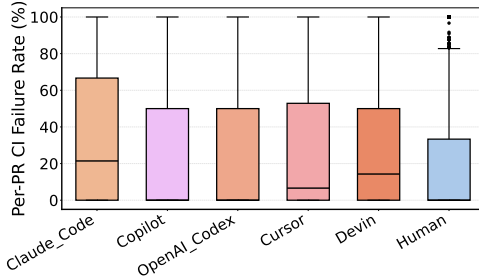


Figure 2: Distribution of build failure rates for AI Agents and human PRs

We compare the two distributions using the Mann–Whitney U test [4], and evaluate the effect size using Cliff’s delta [3]. We first analyze all agentic PRs as a unified cohort, then stratify by AI agent.

Results. agentic-authored PRs exhibit higher CI failure rates than human PRs (27.23% vs. 20.27%). Figure 1 compares the distribution of CI failure rates between human-authored and all agentic PRs, while Figure 2 and Table 1 further break down these failure rates by individual AI agent and report the corresponding statistical comparisons against human PRs. From that Figure, we observe that Agentic PRs exhibit a higher average failure rate with a value of 27.23% compared to human PRs with an average of 20.27%. The Mann–Whitney U test indicates a statistically significant difference ($p < 10^{-29}$), but the effect size is negligible ($\delta = 0.1129$), meaning that while agentic PRs fail more often on average, the difference is modest in practical terms. Moreover, the per AI agent breakdown highlighted in Figure 2 and Table 1, show that Copilot and OpenAI Codex PRs have the lowest failure rates and negligible effect sizes relative to human PRs. Claude Code, Cursor, and Devin exhibit higher failure rates with small effect sizes, indicating modest but more consistent differences in build stability.

Table 1: Comparison of CI failure rates in agentic vs. human PRs.

Group	# PRs	Mean failure rate	p -value	Cliff’s δ
Human PRs	4,152	20.27%	–	–
Claude Code	189	35.48%	8.5×10^{-9}	0.215 (small)
Cursor	630	31.04%	1.3×10^{-13}	0.160 (small)
Devin	2,694	28.89%	1.1×10^{-39}	0.170 (small)
Copilot	1,363	26.58%	7.8×10^{-15}	0.124 (negligible)
OpenAI Codex	2,743	24.47%	7.2×10^{-3}	0.033 (negligible)

3.2 RQ2. Who introduces and repairs CI failures in agentic pull requests?

Approach. We analyze the fail-fix pairs constructed in Section 2.3, which are limited to agentic PRs generated by GitHub Copilot, Cursor, Devin, and Claude Code. For each fail-fix pair, we identify the committer of the failing and fixing commits using repository metadata, following the agent–human attribution rules defined in Section 2.3, and classify responsibility as (fail author $\rightarrow$ fix author). Using this classification, we compute the proportion of CI failures introduced and fixes performed by agents versus humans, both overall and per AI agent type.

Results. In Agentic PRs, AI agents introduce most CI failures (79.15%) but perform a smaller share of the corresponding fixes (60.63%). Table 2 summarizes the distribution of CI failures and fixes across AI agent types, while Table 3 further characterizes the responsibility transitions between AI agent and human committers during CI failure resolution. From Table 2, agents introduce the majority of CI failures with a rate of 79.15%, and their usage to fix these failures slightly reduces to 60.63%. This imbalance indicates that although agents frequently trigger CI failures, human developers remain more involved in repairing them.

Moreover, we breakdown results per each AI agent to analyze the consistency of agents involvement in CI failures and fixes. As shown in Table 2, Devin introduces the highest proportion of failures, accounting for 86.17% of the failures within its fail–fix pairs. Copilot and Cursor introduce CI failures at comparable rates accounting for over 69%, while Claude Code introduces a lower share of failures at 55.77%. Additionally, fix behavior varies where Devin resolves the majority of its introduced failures, fixing 71.76% of them, Copilot resolves roughly half of its failures at 50.22%, while Cursor fixes only 27.16% of the failures it introduces, leaving most fixes to human developers. Across all agents, the proportion of failures fixed by agents is consistently lower than the proportion of failures they introduce, revealing a systematic pattern in which human developers become increasingly involved during failure resolution.

Table 2: AI agent responsibility for CI failures and fixes.

PR Group	# Pairs	Failures by Agents	Fixes by Agents
Claude Code	52	29 (55.77%)	18 (34.62%)
Copilot	462	320 (69.26%)	232 (50.22%)
Cursor	162	113 (69.75%)	44 (27.16%)
Devin	1,041	897 (86.17%)	747 (71.76%)
Overall	1,717	1,359 (79.15%)	1,041 (60.63%)

To further understand repair dynamics, we classify fail-fix pairs by the identity of the failing and fixing committers. As shown in table 3, 22.19% of all *fail* $\rightarrow$ *fix* follow an *agent* $\rightarrow$ *human* pattern, where agents introduce CI failures that are subsequently fixed by humans, compared to 56.96% *agent* $\rightarrow$ *agent* and only 17.18% *human* $\rightarrow$ *human* cases. These results highlight a systematic shift toward increased human involvement during failure resolution, even when failures are predominantly introduced by agents.

Table 3: Distribution of fail-fix responsibility transitions between agents and humans in CI pipelines.

Fail-Fix Type	Count	Percentage
agent→agent	978	56.96%
agent→human	381	22.19%
human→agent	63	3.67%
human→human	295	17.18%

3.3 RQ3. How does the time to fix CI failures differ between AI agent and human commits?

Approach. Using the fail-fix pairs defined in Section 2.3, we measure the time-to-fix as the elapsed time (in minutes) between the start of a failed CI workflow run and the start of its subsequent successful run. Fail-fix pairs are grouped based on the identity of the failing and fixing commit authors (fail author → fix author).

To assess differences in repair speed, we compare time-to-fix distributions across these groups against a human-PR baseline using the Mann-Whitney U test and Cliff’s delta to quantify effect size.

Results. Agentic AI authored fixes resolve CI failures nearly four times faster than human fixes with medians of 17.23 and 71.70 minutes respectively. Figure 3 and Table 4 shows the distributions of time-to-fix across all categories. Failures that are both introduced and fixed by agents are resolved the fastest, with a median time to fix of 17.23 minutes. Failures introduced by humans and subsequently fixed by agents show a similarly low median of 25.20 minutes, indicating that agentic-authored fixes are consistently rapid regardless of who introduces the failure.

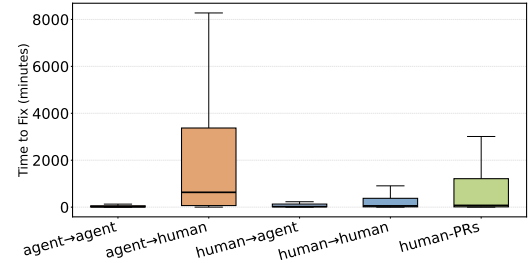
When agents introduce failures but humans perform the fixes, the median time-to-fix increases substantially to 631.57 minutes, whereas failures introduced and fixed by humans have a median resolution time of 53.55 minutes, and human PRs overall exhibit a failure resolution with a median time to fix of 71.70 minutes.

Statistical comparisons against the human-PR baseline confirm these trends. Agent→agent fixes are significantly faster than human PR fixes ($p = 2.70 \times 10^{-42}$, Cliff’s $\delta = -0.319$, small effect). Human→agent fixes are also significantly faster ($p = 4.34 \times 10^{-2}$, $\delta = -0.150$, small effect). In contrast, agent→human fixes are significantly slower than the baseline ($p = 1.17 \times 10^{-19}$, $\delta = 0.299$, small effect), reflecting increased human involvement following agentic-introduced failures. Human→human fixes exhibit negligible differences relative to human PRs overall.

Overall, these results show that agentic-authored fixes are substantially faster than human fixes; however, when agents fail to autonomously resolve CI failures, human developers tend to intervene early in the repair process. For example, in a Copilot-generated PR in kerryjiang/SuperSocket, CI tests repeatedly failed across platforms despite rapid AI agent commits. The maintainer repeatedly requested fixes for persistent test failures on Linux and macOS, and although Copilot responded quickly each time, human guidance was ultimately required to resolve the failures [22]. This aligns with prior findings that developer build fixes incur significant diagnostic and coordination overhead [12], whereas agentic AI can react immediately to CI failures without such delays.

Table 4: Time-to-fix comparison across fail-fix responsibility types. Human PRs serve as the baseline for statistical tests.

Fail-Fix Type	Median (min)	p -value	Cliff’s δ
Human PRs (baseline)	71.70	–	–
agent→agent	17.23	2.70×10^{-42}	–0.319 (small)
human→agent	25.20	4.34×10^{-2}	–0.150 (small)
agent→human	631.57	1.17×10^{-19}	0.299 (small)
human→human	53.55	1.94×10^{-2}	–0.086 (negligible)

**Figure 3: distribution of time to fix CI failures.**

4 Takeaways & Conclusion

This study empirically examined the reliability and maintenance dynamics of agentic software development in terms of their of CI workflows. Two key takeaways emerge from our findings.

(1) **Codex and Copilot PRs exhibit CI failure rates comparable to human PRs.** Looking at RQ1, our results show that agentic PRs, in aggregate, show slightly higher CI failure rates than human PRs, this difference is modest and AI agent dependent. In particular, *OpenAI Codex* and *GitHub Copilot* PRs display failure rates that are statistically indistinguishable from human-authored PRs, with negligible effect sizes. Claude Code, Cursor, and Devin exhibit slightly higher failure rates. This suggests that CI reliability in agentic-assisted development is not uniform and depends on the specific AI agent being used.

(2) **Agents introduce most CI failures, but humans perform a larger share of fixes, even though AI agent fixes are faster.** Our results from RQ2 show that agents introduce the majority CI failures at a rate of 79.15%, and their participation in fixing these failures decreases to 60.63%. At the same time, the analysis from RQ3 reveal that agentic-authored fixes are substantially faster: failures fixed by agents have a median time-to-fix of 17.23 minutes, compared to 53.55 minutes for human-to-human fixes and 71.70 minutes in human PRs overall. This indicates that while agents can efficiently resolve failures, human developers remain heavily involved in the repair process.

These findings highlight a tension in current agentic development practices. Agents are responsible for most failures and fix them faster, yet humans remain more involved in addressing these CI failures. This raises important questions about the current limits of AI agent autonomy: whether agents lack sufficient access to CI execution logs, are constrained by computational or token costs, or cannot accurately retrieve the needed context from the codebase to fix these failures. Understanding and addressing these barriers is critical to moving toward a future in which agentic systems operate 100% on behalf of humans while maintaining trust, reliability, and developer oversight.

References

- [1] 2025. Replication Package. <https://figshare.com/s/3f48566a7e0bef17887c> Figshare. Accessed: 2025-12-29.
- [2] João Helis Bernardo, Daniel Alencar Da Costa, Sérgio Queiroz de Medeiros, and Uirá Kulesza. 2024. How Do Machine Learning Projects Use Continuous Integration Practices? An Empirical Study on GitHub Actions. In *Proceedings of the 21st International Conference on Mining Software Repositories (MSR 2024)*. IEEE/ACM, Lisbon, Portugal, 665–676. doi:10.1145/3643991.3644915
- [3] Norman Cliff. 1993. Dominance statistics: Ordinal analyses to answer ordinal questions. *Psychological Bulletin* 114, 3 (1993), 494–509.
- [4] W. J. Conover. 1999. *Practical Nonparametric Statistics* (3 ed.). Wiley, New York, 608 pages.
- [5] Nicole Forsgren and Jez Humble. 2016. The Role of Continuous Delivery in IT and Organizational Performance. In *Proceedings of the Western Decision Sciences Institute Conference*. Western Decision Sciences Institute, Las Vegas, NV, USA, 15 pages.
- [6] Martin Fowler. 2024. *Continuous Integration*. martinowler.com. <https://martinfowler.com/articles/continuousIntegration.html> Accessed: 2025-XX-XX.
- [7] Taher Ahmed Ghaleb, Daniel Alencar da Costa, Ying Zou, and Ahmed E. Hassan. 2021. Studying the Impact of Noises in Build Breakage Data. *IEEE Transactions on Software Engineering* 47, 9 (2021), 1998–2011. doi:10.1109/TSE.2019.2941880
- [8] GitHub. 2024. GitHub REST API Documentation. <https://docs.github.com/en/rest>. Accessed: Jan. 2025.
- [9] Ahmed E. Hassan, Hao Li, Dayi Lin, Bram Adams, Tse-Hsun Chen, Yutaro Kashiwa, and Dong Qiu. 2025. Agentic Software Engineering: Foundational Pillars and a Research Roadmap. *arXiv preprint arXiv:2509.06216* (2025). doi:10.48550/arXiv.2509.06216
- [10] Junda He, Christoph Treude, and David Lo. 2025. LLM-Based Multi-Agent Systems for Software Engineering: Literature Review, Vision, and the Road Ahead. *ACM Transactions on Software Engineering and Methodology* 34, 5, Article 124 (2025), 30 pages. doi:10.1145/3712003
- [11] Xianhao Jin and Francisco Servant. 2023. HybridCISave: A Combined Build and Test Selection Approach in Continuous Integration. *ACM Transactions on Software Engineering and Methodology* 32, 4, Article 93 (2023), 39 pages. doi:10.1145/3576038
- [12] Noureddine Kerzazi, Foutse Khomh, and Bram Adams. 2014. Why Do Automated Builds Break? An Empirical Study. In *Proceedings of the 2014 IEEE International Conference on Software Maintenance and Evolution (ICSME 2014)*. IEEE, Victoria, BC, Canada, 41–50. doi:10.1109/ICSME.2014.26
- [13] Dipin Khati. 2025. Trustworthiness of Large Language Models for Code. In *Proceedings of the 2025 IEEE/ACM 47th International Conference on Software Engineering - Companion (ICSE Companion)*. IEEE, Lisbon, Portugal.
- [14] Hao Li, Haoxiang Zhang, and Ahmed E. Hassan. 2025. The Rise of AI Teammates in Software Engineering (SE) 3.0: How Autonomous Coding Agents Are Reshaping Software Engineering. *arXiv:2507.15003 [cs.SE]* <https://arxiv.org/abs/2507.15003>
- [15] Kaiyuan Liu, Youcheng Pan, Yang Xiang, Daojing He, Jing Li, Yexing Du, and Tianrun Gao. 2025. ProjectEval: A Benchmark for Programming Agents Automated Evaluation on Project-Level Code Generation. *arXiv preprint arXiv:2503.07010* (2025). doi:10.48550/arXiv.2503.07010
- [16] Microsoft. 2023. Copilot-generated pull request in microsoft/qdk. <https://github.com/microsoft/qdk/pull/2528>. Accessed: 2025-01.
- [17] Ali Ouni, Islem Saidani, Eman Alomar, and Mohamed Wiem Mkaouer. 2023. An empirical study on continuous integration trends, topics and challenges in stack overflow. In *International Conference on Evaluation and Assessment in Software Engineering*. 141–151.
- [18] Thomas Rausch, Waldemar Hummer, Philipp Leitner, and Stefan Schulte. 2017. An Empirical Analysis of Build Failures in the Continuous Integration Workflows of Java-Based Open-Source Software. In *Proceedings of the 14th International Conference on Mining Software Repositories (MSR '17)*. IEEE, Buenos Aires, Argentina, 354–364. doi:10.1109/MSR.2017.54
- [19] Abhik Roychoudhury, Corina Păsăreanu, Michael Pradel, and Baishakhi Ray. 2025. Agentic AI Software Engineers: Programming with Trust. *arXiv preprint* (2025).
- [20] Islem Saidani, Ali Ouni, Moataz Chouchen, and Mohamed Wiem Mkaouer. 2020. Predicting continuous integration build failures using evolutionary search. *Information and Software Technology* 128 (2020), 106392.
- [21] Islem Saidani, Ali Ouni, and Mohamed Wiem Mkaouer. 2021. Detecting continuous integration skip commits using multi-objective evolutionary search. *IEEE Transactions on Software Engineering* 48, 12 (2021), 4873–4891.
- [22] SuperSocket Project. 2023. Pull request #808. <https://github.com/kerryjiang/SuperSocket/pull/808>. Accessed: 2025-01.
- [23] David A. Tomassi, Naji Dmeiri, Yichen Wang, Antara Bhowmick, Yen-Chuan Liu, Premkumar T. Devanbu, Bogdan Vasilescu, and Cindy Rubio-González. 2019. BugSwarm: Mining and Continuously Growing a Dataset of Reproducible Failures and Fixes. In *Proceedings of the 41st International Conference on Software Engineering (ICSE)*. IEEE, Montreal, QC, Canada, 339–349. doi:10.1109/ICSE.2019.00048
- [24] Huaning Wang, Jingzhi Gong, Huawei Zhang, Jie Xu, and Zheng Wang. 2025. AI Agentic Programming: A Survey of Techniques, Challenges, and Opportunities. *arXiv preprint arXiv:2508.11126* (2025). doi:10.48550/arXiv.2508.11126 Version 2.
- [25] 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. *arXiv preprint arXiv:2509.14745* (2025). doi:10.48550/arXiv.2509.14745 Submitted to arXiv.