On the Security of Agentic Pull Requests in GitHub: An Empirical Study

Minh Nguyen Quang Le*, Thanoshan Malavarayar Vijayanandan*, Gouri Ginde, Benjamin Tan, and Lorenzo De Carli

University of Calgary, Calgary, AB, Canada
{minh.le4, thanoshan.vijayanand, gouri.ginde, benjamin.tan1, lorenzo.decarli}@ucalgary.ca
*Equal contribution

Abstract. While prior work has examined the security of AI-generated code at the function-level, a complete understanding of security risks from the use of AI-agents is still missing. We conduct a large-scale empirical study of pull request (PR) security by analyzing 21,454 agentic PRs from the AIDev dataset and 8,103 human PRs manually curated from GitHub, across a wide variety of programming languages. Our results reveal a complex picture. Humans tend to introduce more critical-impact vulnerabilities than AI and Python repositories in both groups exhibit the highest risk. PRs that introduce new features or fix defects are the most vulnerability-prone, and test-related changes in agentic PRs introduce more security issues than in human PRs. Among five coding agents, Claude Code most frequently introduces insecure code, whereas OpenAI Codex shows the highest rate of critical vulnerabilities. Despite these risks, security review engagement remains limited: fewer than 2.2% of comments in vulnerable PRs are security-related, and up to 98.6% of such PRs are merged without security discussion. Notably, CWE-754 (*Improper Check for Unusual Conditions*) is the most common vulnerability type for agentic and human PRs, yet it receives no review attention. These findings have important implications for both practitioners and researchers.

Keywords: AI code security · AI code generation · Software supply chain security

1 Introduction

AI Coding agents (e.g., GitHub Copilot and OpenAI Codex) can assist developers with a wide range of tasks, including generating source code from natural language descriptions, designing software components, creating tests, and fixing bugs [5]. As a result, such agents are widely adopted by developers [22]. However, despite their advantages, the use of AI coding agents also introduces potential security risks. Agents are based on Large Language Models (LLMs) trained on huge amounts of human-written source code that might be unvetted and may contain insecure or flawed implementations. As a result, these models are exposed to software supply-chain risks and may learn to reproduce insecure coding patterns [20].

Although prior work has provided valuable insights into the security of AI-generated code, prior studies focus mainly on vulnerabilities within individual functions [6, 7]. In

practice, however, code does not enter a project as an isolated function: it enters through a PR, which is reviewed, combined, and deployed. The PR is therefore the last point at which a vulnerability can be caught before it reaches production. It is also the point at which the agentic code is currently governed by review practices that were designed for human contributors. Whether those practices still hold is an open question, and answering it requires evidence at the PR level, which remains largely unexplored.

This gap matters for two practical reasons. First, agentic PRs may introduce vulnerabilities in different places than human PRs: different languages, different kinds of change, or different CWE categories. Developers should then focus their review effort and automated checks on those places rather than treating all PRs the same. Second, if reviewers rarely raise security concerns during review, the review stage is not an effective safety net for either group, and adding more agentic code will only enlarge an existing weakness. Therefore, studying the security of AI-generated PRs, and identifying their similarities and differences with respect to human PRs, is essential to reveal patterns of risk, uncover weaknesses in current review practices, and support the safe adoption of AI tools in software development. Specifically, we ask the following research questions:

- *RQ1: What are the rates and CWE patterns of vulnerabilities in agentic PRs, and how do they compare to human PRs?*
- *RQ2: As PRs are expected to undergo public review, how frequently such reviews identify vulnerabilities within?*

To address them, we analyze 21,454 agentic PRs from the AIDev dataset [10], alongside 8,103 human PRs curated from GitHub across repositories with diverse characteristics (we discuss and motivate the design and composition of our dataset in Section 3.1). To assess the presence of vulnerabilities, we use the static analysis tool Semgrep [21]. To categorize PR task types and security-related discussions, we use an LLM-based pipeline. We check the reliability of both against manual annotation (Section 3.3).

Our findings show that human PRs tend to introduce more high-impact vulnerabilities than those generated by AI. Across AI agents, Claude introduces more newly insecure code, while Codex tends to introduce more critical vulnerabilities. Python repositories exhibit the highest rate of vulnerable PR contributions, and PRs aimed at introducing new features or fixing defects are the most likely to contain vulnerabilities. Concerningly, we also find that many PRs are accepted without critical considerations. Only **2.2%** of vulnerable agentic PRs and **1.2%** of vulnerable human PRs receive any security-related review, while up to **96.4%** of vulnerable agentic PRs and **98.6%** of vulnerable human PRs are merged without discussion. Overall, our findings point to important specificities in AI-generated PRs as compared to human-generated ones, and highlight the need for stronger security practices in code review workflows.

In summary, our contributions in this paper are as follows:

- We conduct a large-scale empirical study of PR security introduced by AI and humans.
- We present vulnerability patterns across repository characteristics, PR task types, CWE categories, and five AI coding agents, highlighting differences in security behavior.

- We reveal a critical gap in reviewer security awareness for both agentic and human PRs.
- We curate a real-world human PR dataset from GitHub and publicly release it (https://doi.org/10.5281/zenodo.18463450).

2 Related Work

Applications of AI Agents in Software Engineering. Prior work [19, 4] examined LLM-based coding assistants across a range of programming tasks. Ahmad et al.'s work [1] evaluated ChatGPT for supporting software architecture design, while Sobania et al. [23] and Meng et al. [14] studied the performance of different LLMs in fixing bugs. Our work aims to provide deeper security insights into the use of AI agents by considering its application to PR generation.

Evaluating AI Code Security. Several studies [20, 18] suggest that AI code assistants may have negative security impact on code written by developers. Additionally, Mou et al. [16] assessed LLM code security across tasks such as code completion and vulnerability repair, while Asare et al. [3] conducted an empirical security analysis of Copilot-generated code. However, prior work primarily focus on code snippets. Our PR-level analysis captures the context (e.g., task type, repository features, security discussions), providing a clearer view of how vulnerabilities emerge in practice.

Vulnerability Detection. Several studies focus on detecting vulnerabilities in commit-level patches by leveraging metrics extracted from those changes [11, 24, 17]. Meanwhile, static analysis tools are increasingly being used in research on vulnerability detection [12, 9]. Inspired by these studies, our study assesses security risks in AI-generated PRs.

3 Methodology

Our approach involves detecting newly introduced insecure code using a combination of the static analysis tool and a LLM-based analysis (RQ1, Section 3.2). Further, we use a second LLM-based analysis step to assess reviewers' security awareness (RQ2, Section 3.2). Figure 1 provides an overview of our study.

3.1 Dataset

Our dataset (**Step 1** in Figure 1) combines agentic PRs and human PRs, where human PRs serve as a baseline that reflects non-AI-assisted development. Using both datasets enables that any observed characteristic of AI-generated PR can be contextualized and compared to human-generated ones.

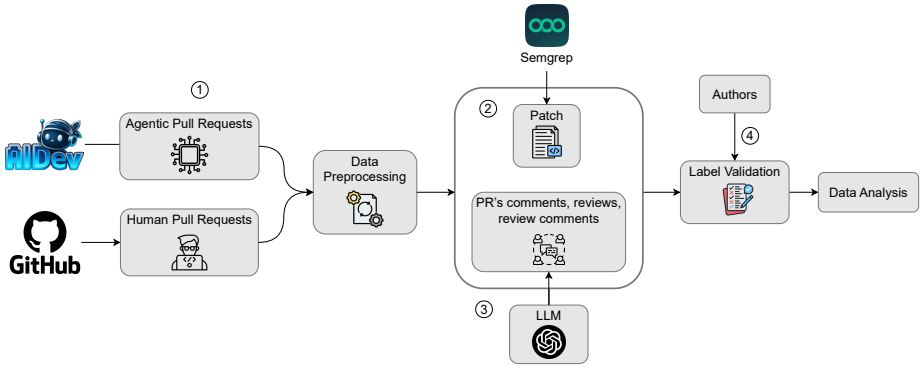


Fig. 1: Study Design Overview

Agentic PR Dataset For agentic PRs, we use seven subsets from the AIDev dataset [10], which is a large-scale dataset containing nearly 1 million PRs created by five major AI agents from over 116,000 open-source repositories. Our analysis is based on the dataset snapshot accessed on November 8, 2025. The seven subsets we use from the AIDev dataset are:

- *pull_request*: contains PR-level metadata.
- *repository*: records repository attributes (e.g., primary language, star count).
- *pr_task_type*: provides task-type labels for each PR.
- *pr_commit_details*: captures file-level metadata included in each commit (e.g., raw patch, diff statistics).
- *pr_review_comments_v2*, *pr_comments*, and *pr_reviews*: includes review discussions, approvals, and feedback exchanged between developers.

Since this study focused on PR security, we analyzed file-level code changes using file patches. After filtering out entries with null patches, we focused on the six most common programming languages in the dataset, as the remaining languages occurred with very low frequency. This process resulted in a dataset of **21,454 agentic PRs** containing files with at least one of the top six file extensions. The distribution of programming languages across the source code files is: `TypeScript` (32.37%), `Python` (17.53%), `TSX` (15.54%), `C#` (13.12%), `Go` (12.38%), and `JavaScript` (9.06%). Among these agentic PRs, OpenAI Codex represents the majority (63.78%), followed by Devin (16.47%), GitHub Copilot (13.51%), Cursor (4.86%), and Claude (1.38%).

Human PR Dataset Although AIDev dataset provides human PRs, its oldest human PR was created on January 2025. As a result, these PRs may include AI-assisted contributions, making it impossible to guarantee a clean human-only dataset. We selected 12,000 PRs between June 2021 to June 2022 by using the GitHub API, restricting the selection to repositories with the same top six programming languages as in the agentic PR dataset. We selected these dates as they pre-date the release of Copilot (the first widespread publicly-available, IDE-integrated AI coding tool), while being recent

enough to mitigate temporal bias. After filtering out PRs that contained null patches, we obtained **8,103 human PRs** with the following language distribution: `C#` (28.65%), `JavaScript` (21.54%), `Python` (18.44%), `TypeScript` (12.14%), `Go` (10.47%), and `TSX` (8.75%).

3.2 Data Analysis

PR Vulnerability Detection. To address RQ1, we aim to identify vulnerabilities newly introduced by a PR (**Step 2** in Figure 1). Accordingly, our approach focuses exclusively on newly added code in each commit of a PR. Specifically, we extracted raw file-level patches and selected the added lines of code. This choice is motivated by prior work [13, 8] showing that the majority of security-relevant changes and vulnerabilities arise from code additions. Thus, we use the term **patch** to refer to all newly added code within a single file.

In order to scan the patches for vulnerabilities, we used the static analysis tool Semgrep (v1.145.2) with the default set of common vulnerability detection rules maintained by Semgrep, including `p/default`, `p/owasp-top-ten`, `p/cwe-top-25`, `p/secrets`, `p/security-audit`, and language-specific rules (`p/typescript`, `p/python`, etc.), running on Ubuntu 24.04. Common rulesets offer similar detection rate across languages, and since the language rulesets focus on specific language features, the detection rate from such rulesets could differ. We performed data validation as described in Section 3.3 and propagated patch-level labels back to their corresponding PRs. Thus, we mark any PR containing at least one vulnerable patch as vulnerable. To validate our approach, we randomly selected and manually inspected 100 PRs marked as vulnerable. In every case, we confirmed the presence of at least one vulnerable patch, providing confidence in the effectiveness of our approach. In total, our analysis scanned **270,384** AI-generated patches across 21,454 agentic PRs and **251,055** human-written patches across 8,103 human PRs.

We further examine which types of development tasks associated with each PR are more likely to introduce new vulnerabilities. Since PR task-type metadata is available only for agentic PRs, we follow the same approach in AIDev dataset [10], using an LLM, to classify the task types of human PRs based on their titles and descriptions. Each PR was assigned to one of the 11 task categories defined by the Conventional Commits Specification (e.g., feat, fix) [10]. This approach enables a consistent comparison of task distributions and vulnerability introduction patterns across agentic and human PRs (we further discuss the choice of LLM and output validation below).

PR Review Analysis. To address RQ2, we filter PRs that contain review records (comments, reviews, or review discussions). We then use an LLM to analyze those threads (**Step 3** in Figure 1). The model was used to determine whether reviewers explicitly identified security issues and whether their discussions referenced the corresponding CWE types. LLMs have been found to be effective in annotating software engineering artifacts [2, 10]. We use OpenAI gpt-5.1-codex-mini, a code-oriented LLM that provides a good balance between annotation accuracy and computational cost. To assess annotation reliability, we validate LLM-generated labels (see Section 3.3); the result indicates

that LLM produces reliable annotations for both PR task types and review discussions. To support reliable annotations, we employ the structured reasoning prompting strategy that has been successful in prior work [25].

3.3 Label Validation

In this section, we evaluate the reliability of Semgrep and the LLM in labeling (**Step 4** in Figure 1). Manual validation was conducted independently by two researchers with software engineering and security background.

Reliability of Semgrep in Vulnerability Detection. We randomly sampled 80 patches labeled as vulnerable and 80 as non-vulnerable from 521,439 patches. For each patch, the annotators inspected the code changes, associated commit messages, the relevant functions before and after the patch, and the affiliated CWE. Then, annotators independently decided whether the patch introduced new vulnerabilities. While initial agreement was low (Cohen's $\kappa = 0.74$), the annotators discussed and resolved each disagreement, refining the annotation guidelines. Finally, the annotators reached consensus labels for all patches, which were used as the ground truth. We then compared with Semgrep's results and achieved 91.25% accuracy, showing that vulnerability labels largely align with human judgment.

LLM Reliability in PR Task and Security Discussion Labels. To assess PR task type classification, we randomly selected 80 human PRs (for agentic PRs, task type labels are available in the AIDev dataset). Annotators independently checked whether each PR was correctly mapped to the corresponding Conventional Commit category [10] as predicted by the LLM, using the same PR metadata given to the model. For security-related review discussion labeling, a random subset of 80 PRs (40 with security-related discussion and 40 without) were selected. Annotators manually inspected each PR's comments, reviews, review comments to determine whether security issues were discussed and the associated CWE(s). The initial inter-rater agreement was $\kappa = 0.71$ for PR task type classification and $\kappa = 0.82$ for security-related discussion labeling. We then followed the same procedure used for validating Semgrep reliability and achieved 87% accuracy for PR task type classification and 85% accuracy for security review discussion labeling. Overall, the LLM-generated labels are sufficiently reliable to support further analyses.

4 Experimental Results

4.1 RQ1: Agentic and Human PR Security Analysis

Among **21,454** agentic PRs and **8,103** human PRs, the vulnerability rates are similar across the two groups: **11.5%** of agentic PRs (2,458 PRs) and **12.8%** of human PRs (1,036 PRs) contained at least one vulnerability.

Repository Language. As shown in Figure 2, both agentic and human PRs exhibit their highest vulnerability rates in Python repositories (31.7% and 36.0%, respectively). This implies that roughly 1 in 3 PRs in repositories introduces a vulnerability. C# repositories show the lowest rates for both groups (1.5% for AI and 2.4% for humans), likely benefiting from strong static typing and stricter compiler checks. The ranking of the next high-risk languages differs between agentic and human contributors. For agentic PRs, it is JavaScript (9.2%), while for human PRs, it is Go (9.3%). Notably, Go repositories show a much lower rate for agentic PRs (3.4%), suggesting that AI agents interact with different language ecosystems in ways that diverge from human developers.

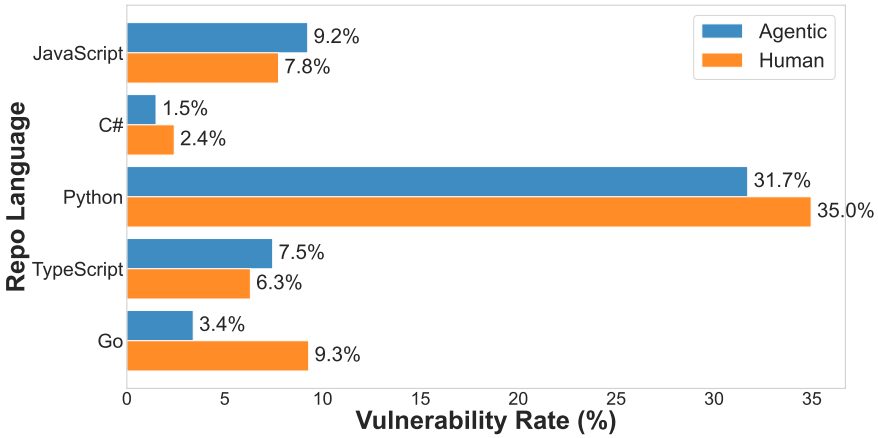
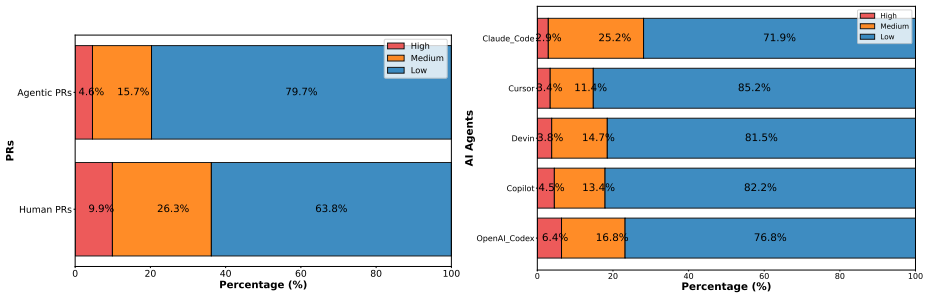


Fig. 2: Vulnerability rate by repository language

PR Task Type. PR tasks were categorized into 11 types according to the Conventional Commits Specification. Among vulnerable PRs, those introducing new features or functionality are the most likely to contain vulnerabilities for both agentic and human contributions (52.0% and 42.8%, respectively). PRs aimed at fixing bugs or defects form the second largest category (23.7% for agentic PRs and 25.0% for human PRs). Together, these results indicate that security risks primarily arise during changes that modify program behavior.

Differences become more apparent in less frequent PR categories. Vulnerable human PRs are more likely to involve non-functional changes such as dependency updates (7.5%) and documentation changes (7.2%), where vulnerabilities may arise from insecure configurations, weak cryptographic recommendations, or unsafe API usage. In contrast, agentic PRs contribute 12.2% of vulnerabilities in test-related code, compared to only 2.2% for human PRs. This indicates that test-related changes, which are often considered low-risk, can still introduce security issues, particularly when generated by AI agents. As a result, security reviews should consider the intent of the PR instead of assuming it is safe.



(a) Severity distribution in human and agentic PRs

(b) Severity distribution in AI agents

Fig. 3: Semgrep severity distributions

Vulnerable Rate By Agents. Table 1 shows that Claude Code is only used for 295 PRs but it produces the highest proportion of insecure code (35.6%). Despite this, only 17.2% of its vulnerable PRs received security attention from the reviewers (see Section 4.2). In contrast, OpenAI Codex contributes the largest absolute number of PRs (1,332) but has a lowest relative vulnerability rate (9.7%).

Table 1: Vulnerable vs. Non-Vulnerable PRs Across Agents

PR Status	**Claude Code**	**Devin**	**Cursor**	**Copilot**	**OpenAI Codex**
Vulnerable PRs	105 (35.6%)	548 (15.5%)	157 (15.1%)	316 (10.9%)	1332 (9.7%)
Non-vulnerable PRs	190 (64.4%)	2985 (84.5%)	885 (84.9%)	2581 (89.1%)	12355 (90.3%)

Severity Level. We analyze the severity levels of identified vulnerabilities as defined by Semgrep rules and associated CWEs. Figure 3a presents a comparison of Semgrep-reported vulnerabilities in human and agentic PRs. Human PRs introduce substantially **more high-severity** (9.9%) and **medium-severity** (26.3%) vulnerabilities than agentic PRs (4.6% and 15.7%, respectively). Conversely, agentic PRs are dominated by low-severity vulnerabilities. This observation aligns with prior work [3], which suggests that AI-generated code tends to introduce vulnerabilities with lower immediate security impact than those introduced by human developers.

Across five AI coding agents, OpenAI Codex tends to introduce the most high-severity vulnerabilities (6.4%), more than twice that of Claude Code (2.9%). This disparity is consistent with our findings in Section 4.1, CWE Characteristics, where OpenAI Codex also produces the largest number of unique CWE types, while Claude Code introduces the fewest. Taken together, these results indicate that agents producing

more diverse or complex code changes are more likely to introduce higher-severity vulnerabilities. By contrast, more conservative agents exhibit narrower CWE coverage and lower-severity vulnerabilities.

CWE Characteristics. In both groups, the top 10 most common CWE types found in the dataset contribute over 92.6% of all vulnerabilities. Among them, 7 CWEs overlap between the two groups, indicating that AI agents and human developers introduce largely similar types of vulnerabilities. However, 5 of the top 10 CWEs in each group do not appear in the latest 2025 CWE Top 25 Most Dangerous Software Weaknesses list [15]. In contrast, CWE-79 (Cross-Site Scripting), identified as one of the most prevalent and impactful weaknesses in the 2025 CWE Top 25, is relatively rare in our findings, accounting for only 1.5% of vulnerabilities in human PRs and 1.0% in agentic PRs. This suggests a gap between the vulnerabilities that occur most often in practice and those emphasized by widely used risk rankings.

Overall, human PRs introduce 50 unique CWE types, while agentic PRs introduce 58. CWE-754 (Improper Check for Unusual Conditions) is the most frequent vulnerability in both agentic (75.1%) and human PRs (57.6%), yet it is absent from the 2025 CWE Top 25. Despite its high prevalence, this CWE receives **no attention** during code review (Section 4.2), revealing a blind spot in current review practices.

Across AI agents, the number of unique CWEs varies. OpenAI Codex introduces the most unique CWEs (48), while Claude Code introduces the fewest (30). This difference might be explained by the fact that OpenAI Codex sees more usage in our dataset than Claude (Table 1), leading to a broader range of observed vulnerabilities. Despite these differences, several CWEs such as CWE-754, CWE-22, and CWE-78 appear across all agents with CWE-754 alone accounting for more than 63% of vulnerabilities for each agent. The analysis reveals that while agents share common vulnerability patterns, they each have unique security blind spots.

4.2 RQ2: Security Review Effectiveness Analysis

Our findings reveal a serious weakness in current code review practices. Reviewer engagement is very limited, particularly for agentic PRs, and security issues are rarely discussed. Among 21,454 agentic PRs, fewer than half (9,568) received any review, and only 453 (4.7%) included vulnerability-related discussion. Human PRs show a similar pattern: only 50 of 5,893 reviewed PRs (0.8%) involved explicit security discussion. Even more concerning, 63.7% of merged agentic PRs and 26.5% of merged human PRs were accepted without any reviewer engagement at all. This means a large amount of code enters repositories with little or no human oversight.

Table 2 highlights an even larger gap. Among 2,458 vulnerable agentic PRs, only **2.2%** contained any vulnerability discussion, and **96.4%** were merged without security-related review. Human PRs show the same trend: just **1.2%** of 1,036 vulnerable PRs involved security discussion, and **98.6%** of merged vulnerable PRs lacked it. In practice, this indicates that security review is the exception, not the norm, and most vulnerable changes are merged with minimal review, increasing the risk of security flaws reaching production.

Table 2: Vulnerability Discussion (VDs) in Vulnerable PRs

PR Status	Vulnerable PRs	Vulnerable PRs with VD	Merged vulnerable PRs	Merged vulnerable PRs without VD
AI	2458	55 (2.2%)	1538	1483 (96.4%)
Human	1036	12 (1.2%)	840	828 (98.6%)

Reviewer attention is highly concentrated on only a few well-known vulnerability types. In both agentic and human PRs, CWE-200 (Exposure of Sensitive Information) is the most frequently discussed issue (11.6% and 15.4%, respectively). CWE-20 (Improper Input Validation) and CWE-79 (Cross-Site Scripting) are also common in agentic PRs, while CWE-20, CWE-798, and CWE-190 appear most in human PRs. Importantly, our RQ1 analysis reveals that CWE-754 (Improper Check for Unusual Conditions) is the **most common vulnerability**, yet it receives **no reviewer attention** in either agentic or human PRs. This suggests that reviewers may focus on familiar and visible problems, while overlooking other frequent vulnerabilities.

Differences also emerge across coding agents. Claude introduces the highest proportion of insecure code (Section 4.1) and also attracts the most security attention, with 17.2% of its PRs receiving security-focused review, while other agents receive less than 7%. In addition, most PRs involve only one or two participants, limiting the diversity of expertise in reviews.

5 Discussion

5.1 Implications

Our findings have implications for both strengthening practitioner workflows and future research. Code reviews poorly reflect actual vulnerabilities, as they rarely align with the vulnerabilities present in PRs. Consequently, studies that rely on review comments to assess PR security may be misleading, and direct analysis of code changes may provide a more reliable assessment. This also points to a systemic gap in review practices. Teams should adopt security-focused review practices into PR workflows to ensure potential issues are addressed before merging. Furthermore, widely used vulnerability rankings may not fully reflect real-world PR security risks. Several CWEs frequently found in PRs are absent from the latest MITRE CWE Top 25 list [15]. Researchers studying PR security should therefore integrate empirical data and/or recent studies. Finally, AI-generated code should be reviewed as carefully as human-generated code, avoiding *automation bias* and presumptions of correctness; as agentic PRs introduce vulnerabilities at rates comparable to human PRs, making it necessary to apply identical security reviews and approval standards to both.

5.2 Threats to Validity

Firstly, we identify newly introduced vulnerabilities by analyzing added lines in PR patches. While this approach is consistent with common static-analysis practices, it may

miss vulnerabilities introduced through code removal (e.g., deleting validation logic). However, as discussed in Section 3.2, the majority of security-relevant changes involve code additions rather than deletions. Secondly, our reliance on Semgrep limits detection to vulnerabilities identifiable without runtime context, and its rule-based nature may cause false positives or missed vulnerabilities. Our LLM-based labeling may produce non-deterministic outputs due to its black-box nature. To mitigate these issues, we manually validated samples of both outputs and applied structured multi-annotator verification. Thirdly, our datasets are imbalanced: the agentic dataset is about 2.6 times larger than the human dataset, and the two differ in language distribution. The two datasets also span different time periods (June 2021 – June 2022 for human PRs vs. 2025 for agentic PRs). We deliberately chose this window to obtain a human baseline free of AI assistance. Finally, our findings are limited to PRs written in the six most common programming languages in the dataset and may not generalize to other languages, ecosystems, or future AI capabilities.

6 Conclusion

This paper presents a large-scale empirical comparison of the security of PRs authored by human developers and AI coding agents. We employed a static analysis tool together with an LLM-based approach to identify, categorize, and analyze vulnerabilities, as well as to examine security-related review discussions. Our findings indicate that while agentic and human PRs exhibit several similar vulnerability patterns, they also differ in the types of weaknesses introduced and the level of attention these issues receive in reviews. Future work should explore hybrid approaches that combine static, learning-based, and dynamic analysis techniques to improve vulnerability detection. Additionally, since different coding agents tend to introduce distinct CWE patterns, further research could investigate these differences and develop targeted strategies to reduce vulnerabilities in AI-assisted development. Overall, our results highlight the need for improved security awareness and effective review practices in modern software development, identifying several research directions for the software security community.

References

1. Ahmad, A., Waseem, M., Liang, P., Fahmideh, M., Aktar, M.S., Mikkonen, T.: Towards human-bot collaborative software architecting with chatgpt. In: EASE (2023)
2. Ahmed, T., Devanbu, P., Treude, C., Pradel, M.: Can llms replace manual annotation of software engineering artifacts? In: 2025 IEEE/ACM 22nd International Conference on Mining Software Repositories (MSR). pp. 526–538. IEEE (2025)
3. Asare, O., Nagappan, M., Asokan, N.: Is github's copilot as bad as humans at introducing vulnerabilities in code? Empirical Software Engineering **28**(6), 129 (2023)
4. Avila-Chauvet, L., Mejía, D., Acosta Quiroz, C.O.: Chatgpt as a support tool for online behavioral task programming. Available at SSRN 4329020 (2023)
5. Bucaioni, A., Ekedahl, H., Helander, V., Nguyen, P.T.: Programming with chatgpt: How far can we go? Machine Learning with Applications **15**, 100526 (2024)
6. Cotroneo, D., de Luca, R., Liguori, P.: Devaic: A tool for security assessment of ai-generated code. Inf. Softw. Technol. **177**, 107572 (2025)

7. Hamer, S., d'Amorim, M., Williams, L.: Just another copy and paste? comparing the security vulnerabilities of chatgpt generated code and stackoverflow answers. IEEE SPW (2024)
8. Iannone, E., Guadagni, R., Ferrucci, F., De Lucia, A., Palomba, F.: The secret life of software vulnerabilities: A large-scale empirical study. IEEE Transactions on Software Engineering **49**(1), 44–63 (2023). https://doi.org/10.1109/TSE.2022.3140868
9. Katz, K., Moshtari, S., Mujhid, I., Mirakhorli, M., Garcia, D.: Siexvults: Sensitive information exposure vulnerability detection system using transformer models and static analysis. arXiv preprint arXiv:2508.19472 (2025)
10. Li, H., Zhang, H., Hassan, A.E.: The rise of ai teammates in software engineering (se) 3.0: How autonomous coding agents are reshaping software engineering (2025), https://arxiv.org/abs/2507.15003
11. Li, Y., Yadavally, A., Zhang, J., Wang, S., Nguyen, T.N.: Commit-level, neural vulnerability detection and assessment. In: Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. pp. 1024–1036 (2023)
12. Li, Z., Dutta, S., Naik, M.: Iris: Llm-assisted static analysis for detecting security vulnerabilities. arXiv preprint arXiv:2405.17238 (2024)
13. Mahyari, A.: A hierarchical deep neural network for detecting lines of codes with vulnerabilities. In: 2022 IEEE 22nd International Conference on Software Quality, Reliability, and Security Companion (QRS-C). pp. 1–7 (2022). https://doi.org/10.1109/QRS-C57518.2022.00011
14. Meng, X., Ma, Z., Gao, P., Peng, C.: An empirical study on llm-based agents for automated bug fixing. arXiv preprint arXiv:2411.10213 (2024)
15. MITRE Corporation: 2025 cwe top 25 most dangerous software weaknesses. https://cwe.mitre.org/top25/archive/2025/2025_cwe_top25.html (2025), accessed: 2025-12-26
16. Mou, Y., Deng, X., Luo, Y., Zhang, S., Ye, W.: Can you really trust code copilots? evaluating large language models from a code security perspective. arXiv preprint arXiv:2505.10494 (2025)
17. Pascarella, L., Palomba, F., Bacchelli, A.: Fine-grained just-in-time defect prediction. Journal of Systems and Software **150**, 22–36 (2019)
18. Pearce, H., Ahmad, B., Tan, B., Dolan-Gavitt, B., Karri, R.: Asleep at the keyboard? assessing the security of github copilot's code contributions. Commun. ACM **68**(2), 96–105 (Jan 2025). https://doi.org/10.1145/3610721, https://doi.org/10.1145/3610721
19. Pirzado, F.A., Ahmed, A., Mendoza-Urdiales, R.A., Terashima-Marin, H.: Navigating the pitfalls: Analyzing the behavior of llms as a coding assistant for computer science students—a systematic review of the literature. IEEE Access **12**, 112605–112625 (2024). https://doi.org/10.1109/ACCESS.2024.3443621
20. Sandoval, G., Pearce, H., Nys, T., Karri, et al.: Lost at c: A user study on the security implications of large language model code assistants. In: USENIX Security (2023)
21. Semgrep, Inc.: Semgrep: Static application security testing platform. https://semgrep.dev/ (2026), accessed: 2026-02-02
22. Shani, I., Staff, G.: Survey reveals ai's impact on the developer experience (Jun 2023), https://github.blog/2023-06-13-survey-reveals-ais-impact-on-the-developer-experience/
23. Sobania, D., Briesch, M., Hanna, C., Petke, J.: An analysis of the automatic bug fixing performance of chatgpt. In: IEEE/ACM APR (2023)
24. Yang, L., Li, X., Yu, Y.: Vuldigger: A just-in-time and cost-aware tool for digging vulnerability-contributing changes. In: GLOBECOM 2017-2017 IEEE Global Communications Conference. pp. 1–7. IEEE (2017)
25. Yu, Z., Guo, Z., Wu, Y., Yu, J., Xu, M., Mu, D., Chen, Y., Xing, X.: Patchagent: A practical program repair agent mimicking human expertise. In: Proceedings of the 34th USENIX Security Symposium (USENIX Security'25), Seattle, WA, USA (2025)