Automated vs. Human Security Patching Patterns in Pull Requests: Evidence from the AIDev Dataset

Felix Wang*
felix.wang@uwaterloo.ca
University of Waterloo
Waterloo, Ontario, Canada

Jacie Jermier*
jacie.jermier@uwaterloo.ca
University of Waterloo
Waterloo, Ontario, Canada

Brian Do*
brian.do@uwaterloo.ca
University of Waterloo
Waterloo, Ontario, Canada

1 Background and Motivation

The rise of autonomous coding agents such as Codex, Cursor, and Claude Code marks a shift toward an era of SE 3.0, where AI systems co-author or independently modify software artifacts [5, 14]. While these agents have boosted developer productivity, their trustworthiness - especially in security contexts - remains underexplored [6, 10, 12]. Existing research has shown differences in review time and merge success between human pull requests (PRs) and agent PRs [1, 7], but there is little systematic evidence on how AI-generated security patches different from human ones at the vulnerability level. Prior research has introduced GraphSPD [13] for identifying security patches among PRs, and TREEVUL [9] trained on human-authored commits that are manually linked to CVEs [3] and annotated with CWE paths. Li et al. demonstrated the applicability of incorporating LLM to categorize security patches on a scope of memory-related vulnerabilities [8].

This study, as the first systematic investigation of the security patch patterns in AI-generated versus human-authored PRs, aims to fill in the gap of automated vulnerability management with empirical study by conducting analysis of security-related PRs: Fully Automated PR, Semi-automated PR, and Fully Human PR, using the AIDev dataset [7]. We distinguish between three types of PRs: A *Fully Automated PR* which is generated and submitted by coding agent; A *Semi-automated PR* which originates from agents but involve human maintainers; and a *Fully Human PR* is entirely finished and created by humans. By mapping these PRs to Common Weakness Enumeration (CWE) [4] tree, we seek to reveal characteristic strengths and weaknesses in AI-generated patches, ultimately informing safer integration of coding agents into modern software development workflows.

2 Research Objectives

This study aims to systematically compare the security-related PRs authored by coding agents and human developers. Specifically, we examine three categories of PRs—Fully Automated, Semi-automated, and Fully Human—to understand their characteristics in vulnerability repair.

RQ1: How does each type of PR's corresponding CWE category distribution differ?

RQ2: Is any type proportionally more prevalent than others? Are there any disproportionate CWE categories in each type of PR?

RQ3: How do different coding agents vary in the types and distributions of CWEs associated with their security-related pull requests?

3 Method

To address the research questions above, we will adopt the AIDev dataset. We will first filter PRs whose titles, descriptions, or comments contain vulnerability-related keywords derived from the list of "2024 CWE Top 25 Most Dangerous Software Weaknesses" [2]. To improve coverage, we will add complementary expressions (e.g., "vulnerab*", "secur*", "privac*") to capture broader security semantics. Incorporating such keyword extraction techniques increases the recall of our results, making the extracted dataset more relevant to security patches.

Prior work shows that LLMs struggle to classify vulnerability types without sufficient contextual information [8]. To address this, we partially adopt their data normalization and augmentation methods to enrich the context for LLM-based categorization. Specifically, we retrieve the raw unified diff files for each PR via the GitHub API (https://patch-diff.githubusercontent.com/raw/[user_name]/[repository_name]/pull/[pr_number].patch), which include both added and removed lines. We then perform function renaming to standardize identifiers and mitigate biases from project-specific or ambiguous names.For example, an unclear function like "put_cred()" is normalized to a semantically descriptive name such as "decrease_credential_reference_count()", a process shown to improve overall classification accuracy [8]. Finally, we apply context slicing with a fixed ±3-line window around each modification to provide additional contextual cues for understanding code semantics.

We selected our LLMs based on two key criteria: reasoning capability and open-source accessibility. Vulnerability classification requires not only pattern recognition but also contextual reasoning-understanding control flow, dependencies, and subtle security implications in code changes. Open-source models, on the other hand, ensure transparency, reproducibility, and the potential for fine-tuning and behavior inspection. Guided by these principles, we chose the top two open-source reasoning models on the GPQA benchmark [11]: *GPT-OSS-120B* and *Nemotron-Ultra-253B*, with 120 billion and 253 billion parameters, respectively. These models exemplify state-of-the-art reasoning performance in complex classification tasks. We employ both in parallel to cross-validate CWE

*All three authors contributed equally to this research.

MSR 2026, Rio de Janeiro, Brazil

ACM ISBN 978-1-4503-XXXX-X/2025/06
https://doi.org/XXXXXXX.XXXXXXX

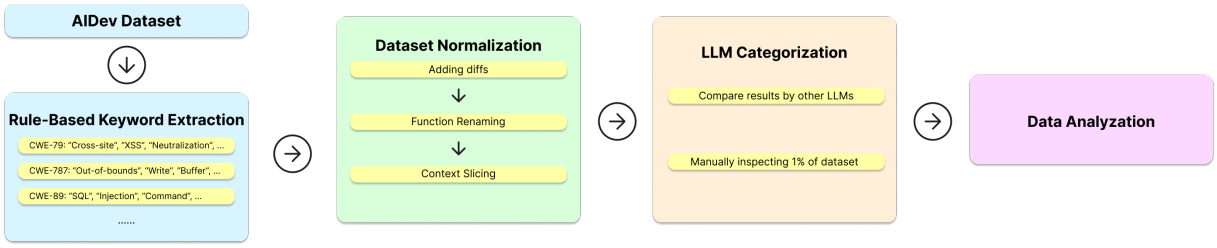

Figure 1: The pipeline of dataset security patch identification.

annotations and reduce labeling noise, followed by a manual inspection of 1% of the dataset to assess classification accuracy.

The prompt we will be using is as follows:

```
You are a professional software developer and system
     security expert.
Decide if the following description of a pull request
     is likely a security patch, then decide its
     related type of Common Weakness Enumeration id.
No explanation is needed, result should be in form of
     {is_security_patch: (boolean), cwes: [{cwe_id,
     cwe_title}]}
Description:
{text}
Code Diff and Context:
{diff_context}
Results:
```

After annotation, we will compare CWE distributions across the three PR categories and across coding agents. Statistical summaries and visualizations will be produced in RStudio to support the analysis of RQ1–RQ3.

4 Milestone schedules

The project timeline spans five weeks from this proposal to the final report submission. Work will be evenly divided into the corresponding five following stages:

- **Keyword Extraction Stage:** Derive keywords for each CWE category from the list of "2024 CWE Top 25 Most Dangerous Software Weaknesses" [2], perform the rule-based keyword extraction procedure, and store the extracted data in Parquet file.
- **Dataset Normalization Stage:** Fetch diff data from the GitHub API endpoint for each record, add two columns of "Renamed function names", and "Sliced Context" for each record, and store normalized data in Parquet file.
- **LLM Categorization Stage:** Run parallel experiments using GPT-OSS-120B and Nemotron-Ultra-253B, and while preparing the analysis scripts for the next phase.
- **Data Analysis Stage:** Conduct manual inspection of 1% of the dataset and compare the output of both LLMs to validate their credibility, and run the scripts to get results and visualizations.
- **Report Preparation Stage:** Compile findings, finalize analyses, and complete the written report for submission.

5 Threats to Validity

5.1 Internal Validity

Keyword Selection Bias. Our initial filtering relies on keywords derived from the "2024 CWE Top 25 Most Dangerous Software Weaknesses" combined with generic vulnerability terms. This approach may over-represent certain vulnerability types while miss less common or subtle cases.

Context Slicing Bias. We extract ±3 lines around the modified line to preserve contextual richness for LLM's categorization. However, this fixed slicing window may fail to capture all relevant control-flow or data-flow dependencies, potentially leading to partial misclassifications.

Manual Inspection Bias. We manually verify 1% of the dataset to assess the credibility of LLM's categorization. While this provides a sanity check, the process is inherently limited by the annotator's expertise and subjective judgment.

5.2 External Validity

AIDev Dataset Bias. The AIDev dataset exhibits significant class imbalance. Among 932,791 agent-involved PRs, 814,522 (87.3%) are attributed to OpenAI Codex, whereas fully human PRs total only 72,189 (7.74%). This over-representation of one agent type limits the generalizability of our findings to other coding agents.

Temporal Limitations. The security patching practices evolve rapidly. Since the AIDev dataset only contains PRs from December 2024 to July 2025, our findings may not fully reflect future developments in agent-assisted coding workflow.

References

[1] Mahmoud Alfadel, Diego Elias Costa, Emad Shihab, and Mouafak Mkhallalati. 2021. On the Use of Dependabot Security Pull Requests. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. 254–265. doi:10.1109/MSR52588.2021.00037

[2] The Mitre Corporation. 2025. 2024 CWE Top 25 Most Dangerous Software Weaknesses. https://cwe.mitre.org/top25/archive/2024/2024_cwe_top25.html

[3] The Mitre Corporation. 2025. CVE Program Mission. https://www.cve.org/

[4] The Mitre Corporation. 2025. CWE- Common Weakness Enumeration. https://cwe.mitre.org/

[5] Ahmed E Hassan, Gustavo A Oliva, Dayi Lin, Boyuan Chen, Zhen Ming, et al. 2024. Towards AI-native software engineering (SE 3.0): A vision and a challenge roadmap. *arXiv preprint arXiv:2410.06107* (2024). https://doi.org/10.48550/arXiv.2410.06107

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

[7] 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

[8] Xingyu Li, Juefei Pu, Yifan Wu, Xiaochen Zou, Shitong Zhu, Qiushi Wu, Zheng Zhang, Joshua Hsu, Yue Dong, Zhiyun Qian, et al. 2025. What Do They Fix? LLM-Aided Categorization of Security Patches for Critical Memory Bugs. *arXiv preprint arXiv:2509.22796* (2025). https://doi.org/10.48550/arXiv.2509.22796

[9] Shengyi Pan, Lingfeng Bao, Xin Xia, David Lo, and Shanping Li. 2023. Fine-grained Commit-level Vulnerability Type Prediction by CWE Tree Structure. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. 957–969. doi:10.1109/ICSE48619.2023.00088

[10] Hammond Pearce, Baleegh Ahmad, Benjamin Tan, Brendan Dolan-Gavitt, and Ramesh Karri. 2025. Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions. *Commun. ACM* 68, 2 (Jan. 2025), 96–105. doi:10.1145/3610721

[11] David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2024. GPQA: A Graduate-Level Google-Proof Q&A Benchmark. In *First Conference on Language Modeling*. https://openreview.net/forum?id=Ti67584b98

[12] Mark Vero, Niels Mündler, Victor Chibotaru, Veselin Raychev, Maximilian Baader, Nikola Jovanović, Jingxuan He, and Martin Vechev. 2025. BaxBench: Can LLMs Generate Correct and Secure Backends?. In *Forty-second International Conference on Machine Learning*. https://openreview.net/forum?id=il3KRr4H9u

[13] Shu Wang, Xinda Wang, Kun Sun, Sushil Jajodia, Haining Wang, and Qi Li. 2023. GraphSPD: Graph-Based Security Patch Detection with Enriched Code Semantics. In *2023 IEEE Symposium on Security and Privacy (SP)*. 2409–2426. doi:10.1109/SP46215.2023.10179479

[14] 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). https://doi.org/10.48550/arXiv.2509.14745