DATA 542 Project - AI Coding Agents Analysis - Final Report

XUANRUI QIU, The University of British Columbia, Canada
JINGTAO YANG, The University of British Columbia, Canada

This project analyzes how five AI coding agents—GitHub Copilot, OpenAI Codex, Devin, Cursor, and Claude Code—generate pull requests in real-world open-source workflows. Using the AIDev dataset, which contains over 33,000 AI-generated PRs and more than 700,000 file-level code changes, we examine three aspects of agent behavior: patch characteristics, description–code semantic consistency, and file-type modification patterns. Our results show clear differences between assistant-style systems (e.g., Copilot) and more autonomous agents (e.g., Devin). Copilot produces small, localized patches, whereas agentic systems submit larger, multi-file changes. We also find that semantic alignment between a PR's description and its code strongly predicts merge success for assistant-style tools but has little effect on agentic systems, suggesting reviewers rely on different evaluation cues depending on patch scale. Finally, our file-type analysis reveals distinct structural patterns in how agents modify production, test, and configuration files. Together, these findings highlight meaningful behavioral differences among AI coding agents and provide insight into how such systems integrate into collaborative software development.

Project GitHub Repo: https://github.com/andrewyang0620/DATA_542_Project_Group_1.git

1 Introduction

AI coding tools are becoming a regular part of modern software development. Developers now rely on systems like GitHub Copilot to write small code snippets, fix bugs, or suggest improvements to their work[7]. At the same time, a newer class of "AI agents", such as Devin and Cursor, has emerged with the goal of performing more complex tasks—generating whole files, running end-to-end workflows, and even submitting full pull requests to a repository[3, 6].

As they become more capable, one important question is how these AI systems actually behave when contributing to real software projects. Do different AI agents write different kinds of patches? Do they describe their changes clearly? And do these behaviors influence how developers review and merge AI-generated PRs?

To explore these questions, we analyze the AIDev[4] dataset from HuggingFace, which contains over 33,000 AI-generated pull requests and more than 700,000 file-level diffs across a variety of repositories. The dataset covers five major AI coding systems—Copilot, OpenAI Codex[2], Devin, Cursor, and Claude Code—making it possible to compare assistant-style tools with more autonomous agents.

Our goal in this project is to understand how these agents differ in their coding behavior and how their PRs are received. By analyzing our research questions, we aim to capture a better picture of how AI systems operate during real development workflows. Our findings help to detect the behavioral differences between assistant-style and agent-style tools, and provide insights into how reviewers interact with AI-generated contributions in open-source projects.

2 Research Questions

Here are our research questions (refined from milestone 1) :

- **RQ1: Patch Characteristics.** How do Agentic-PRs change code, and what are the different characteristics (e.g., additions, deletions, files touched)?
- **RQ2: Semantic Consistency.** How consistent are PR descriptions with the actual code changes?

Authors' Contact Information: Xuanrui Qiu, The University of British Columbia, Canada; Jingtao Yang, The University of British Columbia, Canada.

- **RQ3: File-Type Modification Patterns Across AI Agents.** How does the distribution of modified file types (e.g., production code vs. test files) characterize the structural composition of patches produced by different AI agents?

3 Methodology

3.1 Data Extraction and Joining

We utilize three tables from the AIDev dataset: all_pull_request, pull_request, and pr_commit_details. We select the required columns for analysis. The selected ones are: **pull request metadata** (PR identifier, title, body, state), **agent labels and status** (agent type and merged_at timestamp), and **commit-level diffs** (filename, additions, deletions, and patch text).

The PR-level tables are joined on id, and commit-level details are joined with pull_request.id = pr_commit_details.pr_id.

3.2 Data Cleaning and Wrangling

We apply a multi-step preprocessing pipeline, with data wrangling techniques:

- **Missing values:** PR bodies with null or empty are replaced with 'No description', and a binary variable has_description is constructed.
- **Invalid values:** Rows with missing filenames or missing additions or deletions are removed from pr_commit_details.
- **Consistency:** Only commit-detail rows whose pr_id appears in the main PR table are retained. Duplicate PR identifiers are dropped.
- **Patch metric construction:** For each PR, we compute the sum of additions, sum of deletions, and the number of unique files touched.
- **Outliers:** For each patch metric, we apply an IQR-based rule ($Q3 + 3 \times IQR$) to flag extreme values. Outliers were retained in the analysis to preserve the integrity of the real-world data distribution. However, they are notified and printed out.

3.3 Methodology for RQ1 - Patch Characteristics

We analyze three metrics: total additions, total deletions, and unique files changed. We aggregate file-level differences for every PR to get a PR-level view of patch size. We merge patch metrics with each of the agent labels (Copilot, Devin, OpenAI_Codex, Cursor, Claude_Code). We plot the distribution of each metric across agents using boxplots for visualization to identify patterns in patch size and file change breadth. Because metrics are highly non-normal, we apply the non-parametric Kruskal–Wallis H-test ($p < 0.05$) to assess whether agents differ significantly in patch magnitude.

3.4 Methodology for RQ2 - Semantic Consistency

We check whether the description of a PR (title + body) aligns with the code changes included in the patch[5]. For each PR, we create a natural-language description by combining the title and body together, and a unified code representation by combining all patches associated with the PR. we calculate the cosine similarity between the TF-IDF vectors of the PR description (title + body) and the connected code patch. We use the Mann-Whitney U test to assess if higher consistency correlates with merge success.

3.5 Methodology for RQ3 - File Type Distribution

We classify every filename in the commit details into one of four categories: Production (source code files), Test (files containing "test" or "spec"), Config/Docs (.md, .json, .yaml), and Other. We

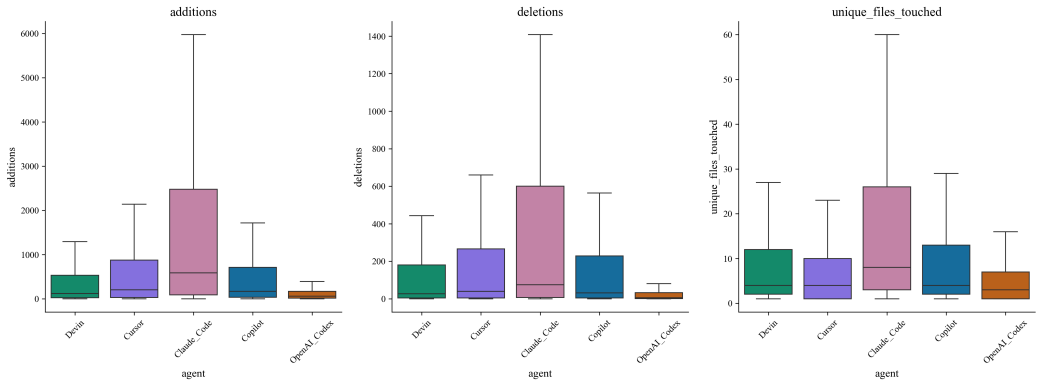

Fig. 1. Distribution of additions, deletions, and unique files touched across AI agents.

apply this classification to all file-level diffs associated with each PR. We then merge the labels with agent information so each file change can be attributed to an AI system. We then compute per-PR counts of how many files of each type are modified and use these counts to assign each PR into a patch-category: Mixed (Code+Test), Production Only, Test Only, or Other. We build a contingency table of agent × patch type and apply a chi-square test to determine whether differences in file-type patterns are statistically significant. Finally, we visualize the distribution using a stacked bar chart showing the percentage breakdown of patch types for each agent.

4 Results (From Compiling the Code)

4.1 Research Question 1

We analyzed patch size characteristics across 33,105 pull requests, summarizing total additions, deletions, and the number of unique files touched per PR. The Kruskal–Wallis H-test indicates statistically significant differences across agents for all three metrics. Specifically, additions show $H = 1608.00$ with p near zero; deletions show $H = 2555.29$ with p near zero; and unique files touched show $H = 686.68$ with $p = 2.67 \times 10^{-147}$. The distribution of these metrics for each agent is visualized in Figure 1. The total time used for running and getting the results is 27.93 seconds.

4.2 Research Question 2

For RQ2, we analyzed semantic similarity between PR descriptions and their corresponding patches using TF–IDF cosine similarity. After merging pull requests with their aggregated patch content, we conducted Mann–Whitney U tests to compare similarity distributions between merged and unmerged PRs for each agent.

The results show that Copilot has a highly significant difference between merged and unmerged PRs ($p = 2.27 \times 10^{-17}$), while OpenAI_Codex also has a weaker but significant difference ($p = 0.0235$). For Claude_Code, Cursor, and Devin, the differences are not statistically significant, with $p = 0.1318$, $p = 0.9048$, and $p = 0.1374$. The distribution of similarity scores for different agents and merge statuses is shown in Figure 2. The total running time is 235.70 seconds.

4.3 Research Question 3

For RQ3, we examined the distribution of file types modified by each AI agent. Each changed file was classified into one of four categories: Production, Test, Config/Docs, or Other, based on filename

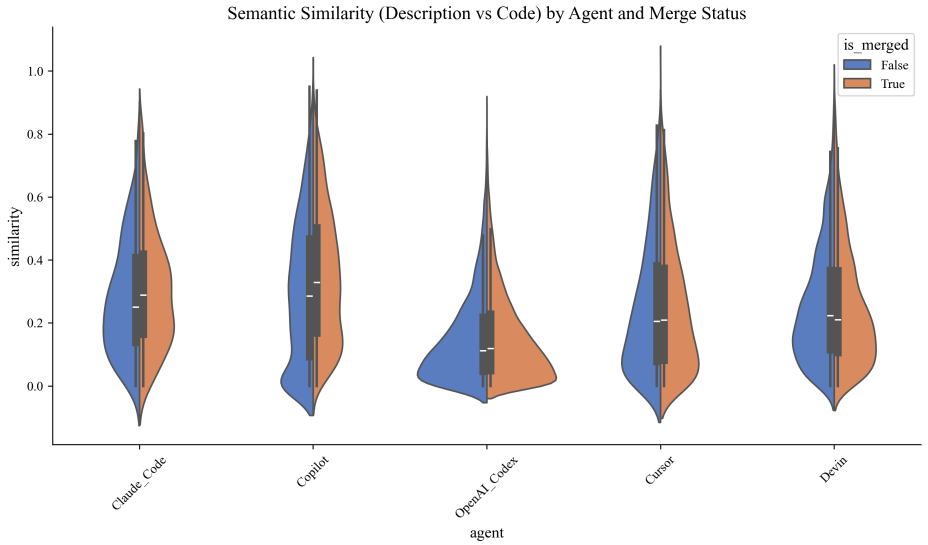

Fig. 2. Semantic similarity between PR description and code patch, grouped by merge status.

patterns. For every PR, we counted how many files of each type were modified and assigned the PR to one of four patch categories: Mixed (Code+Test), Production Only, Test Only, or Other.

The results are as follows:

ClaudeCode: 200 Mixed, 89 Other, 115 Production Only, 53 Test Only

Copilot: 1442 Mixed, 1180 Other, 708 Production Only, 1172 Test Only

Cursor: 365 Mixed, 553 Other, 484 Production Only, 138 Test Only

Devin: 1616 Mixed, 1385 Other, 1420 Production Only, 396 Test Only

OpenAICodex: 7349 Mixed, 5416 Other, 5251 Production Only, 3773 Test Only

A chi-square test of independence showed a highly significant difference in patch-type distributions across the five AI agents, with $p \approx 1.42 \times 10^{-181}$. The percentage breakdown of patch categories for each agent is visualized in Figure 3. The total running time for the RQ3 analysis was approximately 38 seconds.

5 Result Interpretation and Discussion

5.1 Research Question 1

The results from RQ1 show strong differences in patch magnitude for all the five AI agents. Copilot is more likely to produce small patches, which aligns with its character as an assistant-style tool designed for addition types of edits. Devin and OpenAI Codex generate relatively larger patches, which span many more files. This indicates that these two agents operate in a larger scale, which has a more system-level mode. Cursor and Claude Code's output fall between these two , producing moderate patch sizes.

These findings show that agentic systems do not simply scale up assistant-style behavior. They behave qualitatively differently, frequently modifying multiple modules at once. This finding is directly relevant to RQ1.

For our current assessment, a limitation might be that the size of the patch does not reflect the quality of the patch. Large patches may reflect unnecessary over-editing.

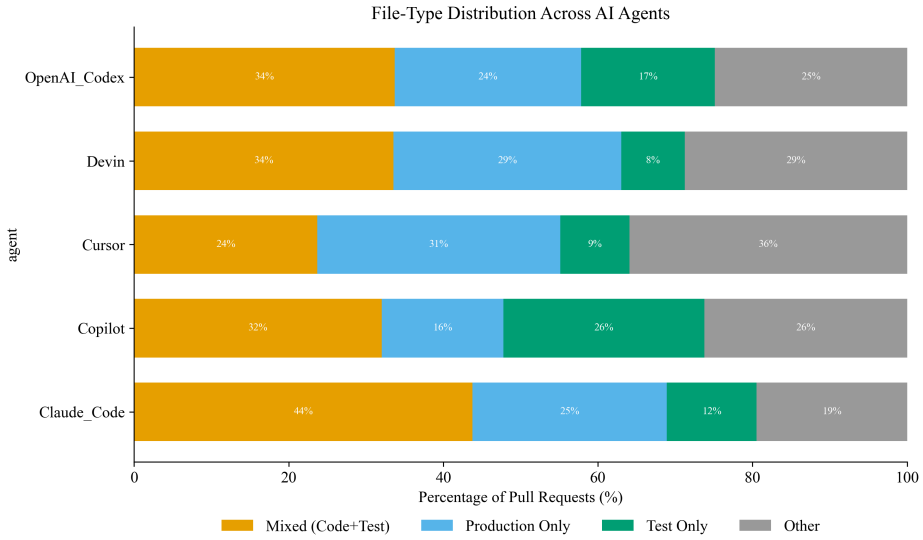

Fig. 3. Distribution of file-type patch categories (Production, Test, Mixed, Other) across the five AI coding agents.

5.2 Research Question 2

The results from RQ2 show a more significant difference on how description–patch similarity relates to merge outcomes for different AI agents. Copilot shows a strong positive association between context similarity and merge success, showing that the textual explanations from reviews is very important when evaluating small, localized edits. OpenAI Codex also shows this effect, only not as significant. However, for Devin, Cursor, and Claude Code, similarity has no significant relationship with merge outcomes. We attribute this shift to the patch magnitude observed in RQ1: Given the complexity of these large, multi-file changes, reviewers likely find high-level descriptions insufficient for verification. Consequently, they may abandon textual alignment as a primary decision standard and instead prioritize functional validation.

These findings highlight a review dynamic difference: assistant tools are evaluated through textual clarity, while agent-style systems are evaluated through functional validation[1].

Limitations include the limited reliance on TF–IDF similarity, which TF-IDF captures surface-level lexical overlap but may underestimate semantic alignment for complex patches or for descriptions written at a higher abstraction level.

5.3 Research Question 3

The results from RQ3 show statistically significant differences in the types of files modified by each AI agent. The result of the chi-square test confirms a strong dependence between the agent and the file-type distribution.

With Copilot, it generates a large proportion of test focused patches, either as test-only changes or as part of mixed patches combining both code and tests. This pattern aligns with its role as an assistant-style tool designed to produce incremental edits. For Devin and OpenAI Codex, they are more likely to produce production oriented or mixed patches. These two agents frequently modify multiple modules and make changes to more substantial portions of the code base. Cursor

contributes to both production and mixed patches but with less emphasis on testing compared to Copilot. Claude Code presents a pattern which focused on producing more production-focused edits than assistant-style tools.

In summary, these results suggest that AI agents differ not only in patch size (as shown in RQ1) but also in the structural composition of the work they produce. Assistant tools like Copilot focuses on smaller-scale edits in test maintenance, whereas agent-style systems tend to assume tasks closer to feature development or system modification. This highlights the broader behavioral divide observed across our analyses.

A limitation is that our file-type classification relies on simple filename rules. This means some projects with unusual folder layouts or non-standard naming conventions may be misclassified, especially when tests or config files are stored in unexpected places.

6 Conclusion

By examining patch characteristics, the semantic alignment between descriptions and code, and the structural composition of modified files, we have found clear behavioral differences between assistant-style AI tools such as Copilot and more autonomous agents like Devin and OpenAI Codex.

A generalized pattern of these tools has been detected across all three research questions: Assistant-style tools such as Copilot produce smaller changes that focus on tests or minor edits, and their merge success is strongly associated with how clearly the PR description matches the underlying code. Agent-style systems generate larger, multi-file patches that behave more like system-level contributions. For these agents, semantic clarity is less important in determining merge outcomes, and reviewers may rely more on functional correctness or other validation signals. Our file-type analysis further reinforces this divide, showing that assistant tools tend to work on incremental or test-related tasks, while agentic systems generate production-level changes.

Together, these findings highlight that AI coding agents not only vary in scale but also differ in the kinds of work they perform and in how their contributions are evaluated by human reviewers. These behavioral differences imply that future code review pipelines, contribution policies, and developer tooling may need to be adapted depending on whether contributions come from assistant-style tools or agentic systems. Future work may address current limitations by employing more expressive semantic representations beyond TF–IDF and adopting richer, project-aware file-type taxonomies, enabling a deeper characterization of agentic coding behavior. Understanding these behavioral distinctions is extremely important for AI agent users, as AI-generated code becomes more common in modern software development workflows.

7 Contributions

- **Xuanrui Qiu**
  - Dataset early investigation and project topic selection
  - RQ2 methodology design and code implementation
  - RQ3 methodology design and code implementation
  - For report, edited the abstract, introduction, and sections related to RQ2 and RQ3
- **Jingtao Yang**
  - Set up GitHub workflows, managed repo updates
  - Design data extraction and cleaning metrics, with code implementation
  - RQ1 methodology design and code implementation
  - For report, edited the conclusion, contribution section, and sections related to data cleaning and RQ1

8 GenAI Disclosure

We used ChatGPT as supportive resources, for the following tasks:

- **Plot styling:** We consulted ChatGPT for suggestions on improving the visualization style used in `plotstyle.py` to produce clearer and more consistent plots.
- **Package usage confirmation:** For the statistical matrices such as Kruskal–Wallis, Mann–Whitney U, chi-square, we used ChatGPT to confirm correct package usage and standard practices.
- **Report writing refinement:** We consulted ChatGPT for proper LaTeX syntax for our report, and used ChatGPT for correcting grammar errors and fine tuning the language of the report.

References

[1] Shraddha Barke, Michael B James, and Nadia Polikarpova. 2023. Grounded copilot: How programmers interact with code-generating models. *Proceedings of the ACM on Programming Languages* 7, OOPSLA1 (2023), 85–111.

[2] Mark Chen. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021).

[3] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2023. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770* (2023).

[4] 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 preprint arXiv:2507.15003* (2025).

[5] Tao Xiao, Hideaki Hata, Christoph Treude, and Kenichi Matsumoto. 2024. Generative AI for pull request descriptions: Adoption, impact, and developer interventions. *Proceedings of the ACM on Software Engineering* 1, FSE (2024), 1043–1065.

[6] Daoguang Zan, Bei Chen, Fengji Zhang, Dianjie Lu, Bingchao Wu, Bei Guan, Wang Yongji, and Jian-Guang Lou. 2023. Large language models meet nl2code: A survey. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 7443–7464.

[7] Zibin Zheng, Kaiwen Ning, Yanlin Wang, Jingwen Zhang, Dewu Zheng, Mingxi Ye, and Jiachi Chen. 2023. A survey of large language models for code: Evolution, benchmarking, and future trends. *arXiv preprint arXiv:2311.10372* (2023).