AgenticFlict: A Large-Scale Dataset of Merge Conflicts in AI Coding Agent Pull Requests on GitHub

Daniel Ogenrwot
University of Nevada Las Vegas
Las Vegas, USA
ogenrwot@unlv.nevada.edu

John Businge
University of Nevada Las Vegas
Las Vegas, USA
john.businge@unlv.edu

Abstract

Software Engineering 3.0 marks a paradigm shift in software development, in which AI coding agents are no longer just assistive tools but active contributors. While prior empirical studies have examined productivity gains and acceptance patterns in AI-assisted development, the challenges associated with integrating agent-generated contributions remain less understood. In particular, **merge conflicts**, a fundamental aspect of collaborative software development, remain underexplored in this context. In this paper, we present AgenticFlict, a large-scale dataset of textual merge conflicts in AI coding agent pull requests (Agentic PRs). The dataset comprises 142K+ Agentic PRs collected from 59K+ repositories, of which 107K+ are successfully processed through deterministic merge simulation. Our pipeline identifies 29K+ PRs exhibiting merge conflicts, yielding a conflict rate of 27.67%, and extracts 336K+ fine-grained conflict regions across these instances. Our preliminary exploratory analysis indicates that merge conflicts are both frequent and often substantial in AI-generated contributions, with noticeable variation across agents, emphasizing the need to better understand and manage integration challenges in AI-assisted software development. **The dataset, code and supplementary materials are available in zenodo: 10.5281/zenodo.19396916**

CCS Concepts

• **Software and its engineering** → **Software creation and management**; **Empirical software validation**; **Collaboration in software development**; • **Computing methodologies** → **Artificial intelligence**.

Keywords

AI coding agents, Agentic AI, Merge Conflicts, Pull Requests, AIDev

ACM Reference Format:
Daniel Ogenrwot and John Businge. 2026. AgenticFlict: A Large-Scale Dataset of Merge Conflicts in AI Coding Agent Pull Requests on GitHub. In *Proceedings of the 3rd ACM International Conference on AI-Powered Software (AIware '26), July 6–7, 2026, Montreal, QC, Canada.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3805760.3814923

1 Introduction

The rise of Artificial intelligence (AI) coding agents is reshaping modern software development workflows. Several AI coding tools such as GitHub Copilot [20], OpenAI Codex [43], Claude Code [3], Cursor [15], and Devin [16] assist developers by generating code, suggesting refactorings, and increasingly contributing changes in the form of pull requests (PRs). This evolution reflects a broader shift from assistive tooling toward active collaboration, often described as Software Engineering 3.0 [29, 30, 41]. Prior work has examined how developers interact with AI-generated code and the impact of these tools on productivity and software quality [34, 37, 38, 52, 53]. These studies highlight both the opportunities and challenges associated with human–AI collaboration. More recent empirical research has begun to investigate development efficiency and code review dynamics in AI-assisted settings [38, 56, 59]. However, a fundamental aspect of collaborative software engineering, namely **merge conflicts**, remains largely unexplored in the context of AI-generated contributions.

In practice, integrating code in a collaborative environment is rarely smooth due to conflicts [7, 8, 10, 39, 48]. Merge conflicts arise when concurrent modifications affect overlapping regions of code and cannot be automatically reconciled by version control systems like Git. Prior research has shown that merge conflicts introduce substantial coordination overhead and negatively impact developer productivity [6, 28, 36, 44, 55]. Brun et al. [6] demonstrate that collaboration conflicts are frequent and costly in distributed development environments. Gousios et al. [27] analyze pull-based development workflows and highlight the complexity of integrating contributions through PRs. Studies on modern code review further emphasize that integration friction influences review latency and decision making [33, 57].

Despite the growth of empirical research in this area, curated datasets specifically targeting merge conflicts remain limited. While Shen and Meng [49] introduced ConflictBench as a dedicated benchmark for studying conflicts, other existing datasets have typically emerged as secondary artifacts of broader studies on collaborative development [11, 18, 51]. More recently, Li et al. [34] presented AIDev, a large-scale dataset capturing PRs (*a.k.a* Agentic PRs), issues, and discussions involving five AI coding agents. However, these datasets fail to provide explicit, reproducible labels for textual merge conflicts; instead, they prioritize metrics such as acceptance rates, temporal dynamics, and general repository characteristics. Watanabe et al. [57] report that merge conflicts account for over 1.1% of Agentic-PR rejections. A concrete example is observed in openai/codex-PR#612 , where the pull request was abandoned because the contributor was unable to resolve the merge conflict.

Researchers currently lack the necessary resources to study integration friction introduced by AI coding agents in collaborative software engineering environments. To address this gap, we introduce AgenticFlict, a large-scale dataset of textual merge conflicts

This work is licensed under a Creative Commons Attribution 4.0 International License.
AIware '26, Montreal, QC, Canada
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2601-9/2026/07
https://doi.org/10.1145/3805760.3814923

in AI coding agent PRs derived from AIDev dataset [34]. The dataset comprises 142,652 Agentic PRs collected from 59,412 repositories, of which 107,026 are successfully processed through deterministic merge simulation of open and/or closed (unmerged) PRs. Our pipeline identifies 29,609 PRs exhibiting merge conflicts, yielding a conflict rate of 27.67%, and extracts 336,380 fine-grained conflict regions across these instances. Beyond binary conflict labels, AgenticFlict provides detailed conflict-region metadata, including affected file paths, conflict regions, and line-level spans. The dataset spans contributions from five distinct AI coding agents, enabling comparative analysis of conflict prevalence and severity across agents.

To the best of our knowledge, AgenticFlict is the first large-scale dataset of textual merge conflicts in Agentic PRs. This dataset can support several research directions, including: (i) empirical studies of merge conflict prevalence and characteristics in AI-generated code; (ii) comparative analysis of integration behavior across AI coding agents; (iii) training and evaluation of automated conflict detection and resolution models; (iv) analysis of the relationship between pull request characteristics (e.g., size, files changed) and conflict likelihood or severity; and (v) benchmarking tools for conflict prediction, merge automation, and collaborative development support in AI-assisted workflows.

In summary, the **contributions** of this work are as follows:

- A reproducible merge simulation pipeline for large-scale conflict detection in pull requests.
- A pull request level dataset containing textual conflict labels and severity metrics.
- A fine-grained conflict-region dataset with file paths and exact line spans of conflicting regions.
- A publicly released artifact to support research on AI-assisted collaboration and integration friction.

2 Dataset Curation Methodology

Figure 1 summarizes the multi-stage workflow used to construct the AgenticFlict dataset. The pipeline consists of five main stages: (1) Agentic PR collection from the AIDev dataset, (2) Metadata Retrieval, (3) repository preparation, (4) deterministic merge simulation, and (5) conflict extraction.

Step 1: Pull Request Collection. We use the AIDev dataset [34] downloaded from Hugging Face as of January 5, 2026. The dataset contains 932,791 Agentic PRs. As an initial preprocessing step, we retain PRs that are either open or closed without evidence of having been merged. When raw state and merge timestamps are available, this filtering is performed before the extraction pipeline begins; otherwise, the final decision is deferred to GitHub metadata retrieval in next step of the pipeline. This filtering yielded **142,652** candidate PRs. Although this design may slightly reduce dataset coverage, it guarantees that all retained records correspond to verifiable GitHub artifacts. Each pull request is identified by a repository name (`repo_full_name`), in the form (`owner/repository`) and a pull request number (`pr_number`). We combine these to construct a canonical identifier (`pr_key`) of the form `repo_full_name#pr_number`, enabling consistent tracking throughout the pipeline.

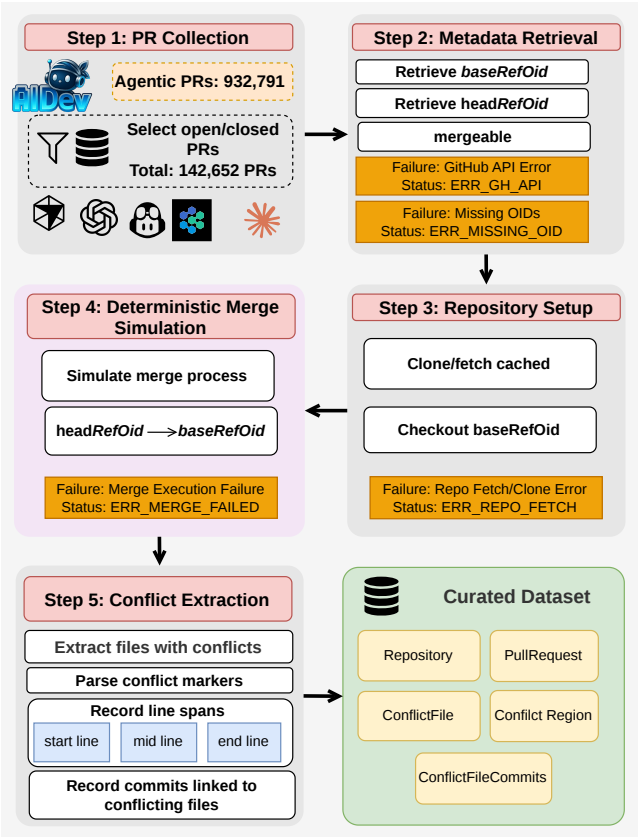

Figure 1: Overview of the AgenticFlict dataset curation workflow.

Step 2: Metadata Retrieval. For each pull request, we query the GitHub GraphQL API [22] to retrieve repository and pull request metadata, including the pull request state, timestamps, branch names, and the base and head commit object identifiers (`baseRefOid` and `headRefOid`), which serve as the primary anchors for merge simulation. At scale, interacting with the GitHub API introduces several practical limitations. In particular, requests may fail due to rate limiting (HTTP 403) [13, 23, 32], transient server errors (e.g., HTTP 502/503), or repository-level issues such as deletion or restricted access (HTTP 404/410/451) [13, 32]. To mitigate these challenges, our implementation employs bounded retries with increasing delays to mitigate transient API failures, token rotation to distribute request load, and explicit handling of API error codes. Despite these safeguards, some PRs remain unrecoverable due to permanently missing references or inaccessible repositories. We identified 35,626 such cases. Instead of silently discarding them, we explicitly record failure modes using structured status codes and exclude these instances from downstream conflict analysis.

Step 3: Repository Preparation. Before merge simulation, each repository is prepared locally. We clone repositories into a persistent cache using Git's partial clone mechanism [19], which downloads repository history while avoiding unnecessary file blobs. Subsequent PRs belonging to the same repository reuse the cached clone

and perform a lightweight `git fetch` to synchronize the repository state.

For each pull request, the pipeline resets the working tree to a clean state and checks out the base commit identified by `baseRefOid`. This preparation step ensures that every merge simulation begins from a deterministic repository state and avoids interference from previous operations.

Step 4: Deterministic Merge Simulation. Algorithm 1 summarizes this step. To determine whether a pull request produces a textual merge conflict, we perform a local merge simulation using Git. Given the base and head commit OIDs retrieved from GitHub, we execute the following command: `git merge -no-commit -no-ff <headRefOid>`. If the merge completes successfully, the pull request is labeled `merge_clean`. If the merge fails, the repository enters a conflicted state and we proceed to conflict extraction.

The simulation procedure differs slightly depending on pull request state. For open PRs, we simulate the merge using the current base and head commit OIDs returned by the API. For closed but unmerged PRs, the base branch may have advanced after closure; therefore, we reconstruct the base commit corresponding to the repository state at the time the pull request was closed and perform the merge against that snapshot. This design approximates the merge conditions that developers would have encountered at closure time.

Certain merge simulation failures may arise when repositories have been deleted or historical commits are no longer reachable due to force-pushes or history rewrites [4, 9, 47]. Such cases are explicitly labeled with structured error codes and recorded in the dataset's run log, allowing downstream analyses to quantify merge simulation coverage.

Step 5: Conflict Detection and Region Extraction. When a merge operation fails, Git records unresolved files in the index. As indicated in Line 9 of the Algorithm 1, we identify these files using: `git diff -name-only -diff-filter=U`. Each conflicted file contains standard Git conflict markers: <<<<<<<, =======, and >>>>>>>. We parse these markers to extract structured conflict regions. For each region, we record several parameters including: file path, conflict index within the file, line boundaries (`start_line`, `mid_line`, `end_line`), SHA-256 hashes of each side's code block, and short textual previews of each side. In addition, we also compute PR-level severity metrics such as the number of conflicting files, number of conflict regions, and total number of lines contained within conflict markers.

To balance dataset size and comply with repository licensing constraints, we store compact representations of conflicts, including content hashes and short previews (default: 5 lines of code), rather than full conflict blocks. This approach follows established practices in large-scale mining of GitHub data [17, 26, 50] and aligns with GitHub's Terms of Service governing code redistribution [21].

Beyond identifying conflict regions, we attribute each conflicting file to the most recent commit that modified the file on both the base and head sides of the merge. This attribution is computed using: `git log -n 1 -format=%H <rev> - <file>`. The resulting fields `head_last_touch_oid` and `base_last_touch_oid` provide a lightweight proxy for identifying the commits most directly associated with the conflicting file. Although this approach does not perform line-level blame alignment, it provides sufficient granularity for studying commit structuring, change locality, and conflict concentration.

Algorithm 1: Deterministic Merge Simulation and Conflict Extraction

Input: Repository path R, base commit b, head commit h
Output: Merge outcome, conflict metrics, and extracted conflict regions

1 Reset the working state of repository R;
2 Checkout base commit b;
3 Create a temporary analysis branch;
4 $rc \leftarrow$ SimulateMerge(R, h);
5 **if** rc = *Success* **then**
6 Revert temporary merge state;
7 **return** `merge_clean`, empty metrics, empty region set;
8 $F \leftarrow$ ListConflictedFiles(R);
9 $Regions \leftarrow \emptyset$;
10 Initialize metrics:;
11 $num_conflict_files \leftarrow 0$;
12 $num_conflict_regions \leftarrow 0$;
13 $conflict_lines \leftarrow 0$;
14 **foreach** $f \in F$ **do**
15 $text \leftarrow$ ReadFile(R, f);
16 $R_f \leftarrow$ ParseConflictRegions($text$);
17 Add R_f to $Regions$;
18 Update metrics using the extracted regions from R_f;
19 Revert temporary merge state;
20 **return** `merge_conflict`, metrics, $Regions$;

3 Dataset Schema and Dataset Overview

The dataset is organized as a relational schema supporting analysis at multiple levels of granularity. We provide both a raw dataset, which includes full pipeline metadata, and a clean dataset, which retains only analysis-relevant attributes. The discussions and results in this paper are based on the clean dataset. A detailed mapping of retained and removed fields is included in the replication package [40].

3.1 Schema Overview

We describe the schema of AgenticFlict, illustrated in Figure 2. The schema consists of five primary entities, which are explained below. Additional details on field definitions are provided in the replication package as an online appendix [40].

Repository. The repository entity stores contextual metadata about repositories referenced in the dataset, including repository name, star count, fork count, primary programming language, and repository status (e.g., archived or fork). Separating this information avoids redundancy when multiple PRs originate from the same repository.

PullRequest. The PR entity is the central component of the dataset and contains one record per pull request. It stores GitHub metadata such as repository identifier, pull request number, state, timestamps,

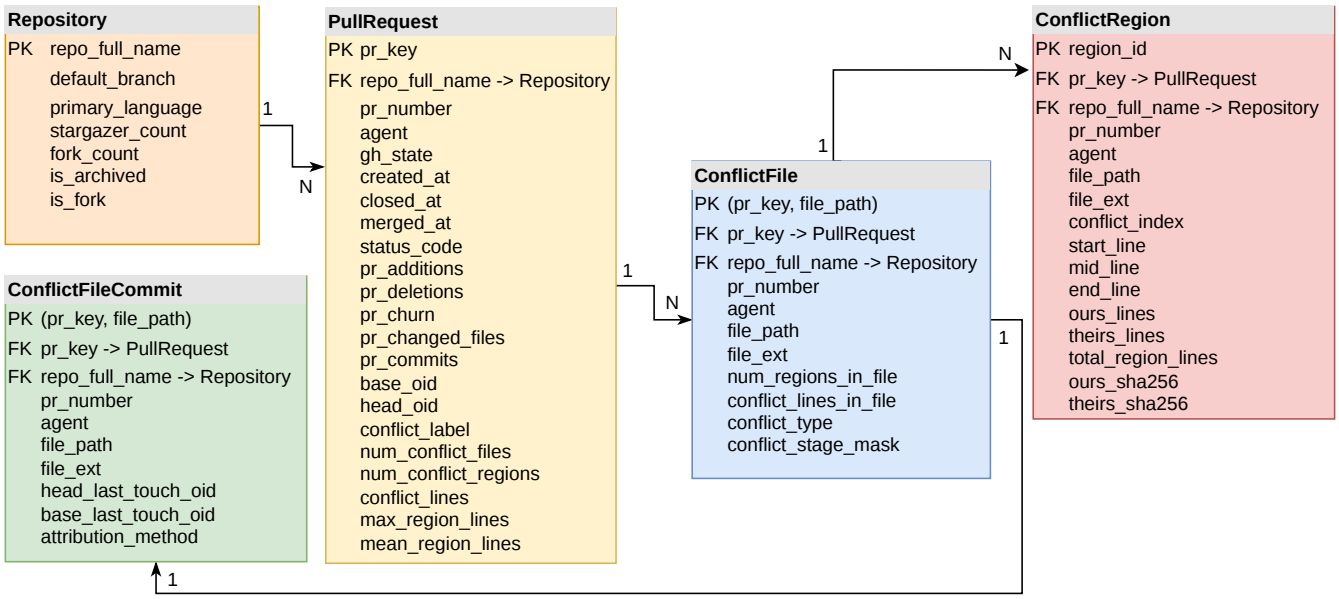

Figure 2: Dataset Schema of AgenticFlict.

and mergeability signals. In addition, it records reconstruction outcomes, including whether a conflict occurs and aggregate severity metrics such as the number of conflicting files, number of conflict regions, and total conflict lines.

ConflictFile. This entity captures file-level conflict information and is linked to the pull request entity via `pr_key`. Each record corresponds to a file containing at least one conflict region and includes attributes such as the number of conflict regions, total conflict lines, file extension, and conflict type (e.g., both-modified, modify/delete).

ConflictRegion. Provides fine-grained conflict details. Each record represents a single conflict region within a file and includes the file path, region index, and line-level boundaries (`start_line`, `mid_line`, `end_line`). Additional attributes capture the size of each side of the conflict and compact hash representations of the conflicting code blocks.

ConflictFileCommit. This entity links conflicting files to the commits most recently modifying them on each side of the merge. For each conflicting file, we record the last commit touching the file on the head and base branches. This provides a lightweight approximation of the origins of conflicting changes and enables analyses of conflict provenance.

3.2 Dataset Overview

Table 1 provides an overview of the AgenticFlict dataset. Starting from 142,652 Agentic PRS, we successfully performed merge simulation for 107,026 instances, corresponding to a success rate of 75.03%. The remaining PRs were excluded due to missing commit references or repository access limitations, which are common challenges when working with large-scale GitHub data [4, 32, 46].

Among the successfully simulated PRs, we observe that merge conflicts are relatively frequent. In particular, 27.67% of PRs result in textual conflicts, indicating that integration issues are not uncommon in AI-generated contributions. This suggests that, despite their usefulness, AI coding agents can introduce non-trivial challenges during code integration.

We further examine the severity of these conflicts by focusing on PRs that exhibit conflicts. On average, a conflicting pull request affects 4.36 files, with a median of 2 files, indicating that most conflicts are relatively localized, but a subset involves multiple files. Each conflicting pull request contains an average of 11.36 conflict regions and over 500 conflicting lines, suggesting that conflicts are often substantial rather than isolated. Overall, the dataset contains more than 336,000 fine-grained conflict regions.

Finally, the dataset spans 59,412 distinct repositories and includes contributions from five different AI coding agents. This diversity provides a broad view of how Agentic PRS behave across different projects and development contexts, supporting comparative and large-scale empirical analyses.

4 Exploratory Empirical Analysis

We perform an exploratory empirical analysis to characterize merge conflict behavior in Agentic PRS. Specifically, we investigate (1) how pull request size relates to the likelihood of merge conflicts, and (2) how conflict rates and severity vary across different AI coding agents. These analyses provide initial evidence on how change characteristics and agent behavior influence integration outcomes in AI-assisted software development.

❓ How do merge conflict rates and severity vary across AI Coding Agents? To understand whether different AI coding agents exhibit distinct integration behaviors, we analyze conflict rates and conflict severity across agents.

Table 1: Summary statistics of the AgenticFlict dataset.

Metric	Value
Dataset Overview	
Total AI PRs identified	142,652
Valid PRs with identifiers	142,652
Successfully simulated PRs	107,026
Excluded PRs	35,626
Merge simulation success rate	75.03%
Merge Outcomes	
Conflicting PRs	29,609
Clean PRs	75,924
Conflict rate	27.67%
Conflict Severity (per conflicting PR)	
Mean conflicting files per PR	4.36
Median conflicting files per PR	2.00
Mean conflict regions per PR	11.36
Mean conflict lines per PR	540.42
Total conflict regions	336,380
Dataset Diversity	
Distinct repositories	59,412
Distinct AI agents	5

Table 2 summarizes the number of PRs, conflicting PRs, and corresponding conflict rates with 95% confidence intervals for each agent. We observe substantial variation across agents. Copilot exhibits the lowest conflict rate at 15.43%, followed by Cursor (20.06%) and Devin (23.04%). In contrast, OpenAI Codex shows the highest conflict rate at 32.31%, more than double that of Copilot. Claude_Code also demonstrates relatively high conflict rates (26.86%), although with wider confidence intervals due to smaller sample size.

Figure 3 visualizes these differences with confidence intervals, highlighting that the variation is statistically meaningful. The separation between agents, particularly between Copilot and OpenAI Codex, suggests that the likelihood of introducing merge conflicts varies significantly depending on the underlying AI system.

To further examine the nature of these conflicts, we analyze conflict severity, measured as the number of conflicting lines per PR. Figure 4 shows the distribution of conflict severity across agents on a logarithmic scale. We observe heavy-tailed distributions for all agents, indicating that while most conflicts are relatively small, some PRs introduce very large and complex conflicts. Notably, OpenAI Codex exhibits both a higher conflict rate and a broader spread of conflict severity, suggesting that it not only conflicts more frequently but may also produce more complex integration challenges.

Overall, these findings indicate that AI coding agents differ not only in how often they produce conflicting changes, but also in the magnitude of those conflicts. This highlights the importance of considering agent-specific behaviors when designing integration workflows and evaluation benchmarks for AI-assisted software development.

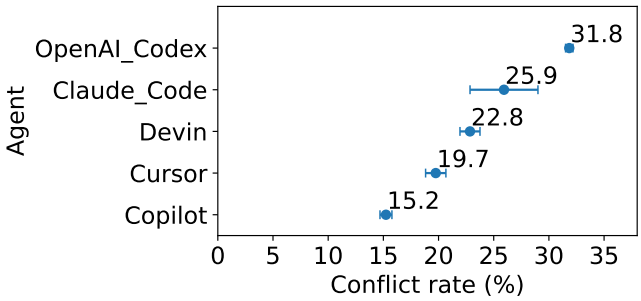

Figure 3: Conflict rates across AI coding agents with 95% confidence intervals.

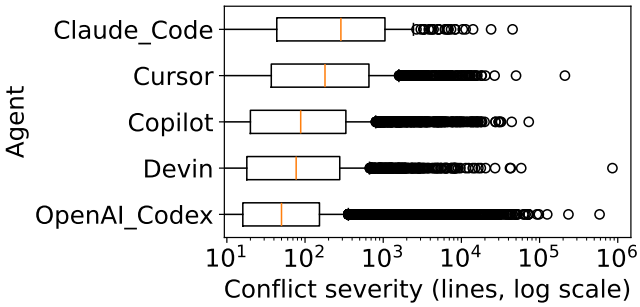

Figure 4: Distribution of conflict severity (measured as conflicting lines) across AI coding agents.

> **Key takeaway:** AI coding agents differ in both the frequency and severity of merge conflicts, highlighting the need for agent-aware integration workflows and evaluation strategies.

How does pull request size affect merge conflict likelihood? We investigate the relationship between PR size and the likelihood of merge conflicts. We measure PR size using *code churn*, defined as the sum of lines added and deleted in a pull request. To analyze this relationship, we group PRs into deciles based on churn and compute the conflict rate within each bin.

Figure 5 shows the resulting relationship between PR size and conflict rate. We observe a clear trend: smaller PRs are significantly less likely to exhibit merge conflicts, while conflict rates increase rapidly as PR size grows. For example, PRs with a median churn of 2 lines have a conflict rate of approximately 9.9%, whereas PRs with a median churn of 25 lines exhibit a conflict rate of nearly 30%.

The conflict rate continues to increase and stabilizes around 32–33% for medium-sized PRs (median churn between 46 and 185 lines). For larger PRs, the conflict rate slightly decreases but remains substantially higher than that of small PRs, suggesting that large changes consistently introduce higher integration complexity.

These findings indicate that integration difficulty is associated with the size of AI-generated changes. Larger PRs are more likely to interfere with concurrent development activity, leading to a higher probability of textual merge conflicts. This finding highlights the importance of controlling change size in AI-assisted development workflows to reduce integration friction.

Table 2: Conflict rates across AI coding agents with 95% confidence intervals.

Agent	PRs	Conflicting PRs	Conflict Rate (%)	95% CI Low	95% CI High
Copilot	16954	2583	15.24	14.69	15.78
Cursor	7196	1421	19.75	18.83	20.67
Devin	8241	1883	22.85	21.94	23.76
Claude_Code	779	202	25.93	22.85	29.01
OpenAI_Codex	73856	23520	31.85	31.51	32.18

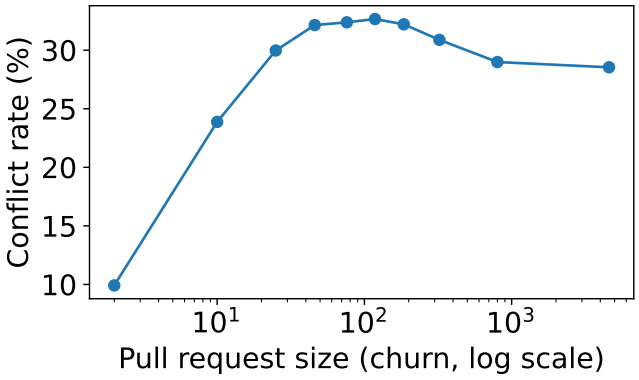

Figure 5: Conflict rate as a function of pull request size (measured as code churn). PRs are grouped into deciles based on size.

> **Key takeaway:** Integration difficulty increases with the size of AI-generated changes, as larger PRs are more prone to merge conflicts.

5 Related Work

Pull-Based Development and Code Review. Pull-based development has become the dominant contribution model in open source ecosystems. Gousios et al. [27] conducted one of the first large-scale empirical studies of the pull-based model, analyzing review practices, acceptance rates, and integration dynamics. Later work examined review quality, reviewer behavior, and factors influencing pull request acceptance [2, 24, 25, 33]. While these studies provide valuable insight into collaborative workflows, they typically do not reconstruct merge outcomes at the commit level. As a result, integration friction due to textual conflicts is not explicitly captured in most pull request datasets.

AgenticFlict complements prior PR research by introducing conflict-aware metadata that can be integrated with review and acceptance analyses, filling a critical gap in understanding how modern automated and agentic contributions impact repository health [57].

AI-Assisted and AI-Generated Code Contributions. The emergence of large language models for code generation has motivated empirical research on AI-assisted programming. Controlled experiments show that developers complete tasks faster when assisted by systems such as GitHub Copilot [45]. Human-computer interaction studies examine developer expectations and usability challenges of code generation tools [53]. More recently, large-scale mining studies have begun to analyze repositories containing AI-generated or AI-assisted contributions [31, 34, 41, 57]. These studies examine acceptance rates, code quality, and maintenance characteristics. However, they do not explicitly study merge outcomes or quantify textual conflict severity.

Our work provides a large-scale dataset of reproducible textual merge conflict labels and fine-grained conflict-region metadata for Agentic PRs.

Merge Conflicts in Collaborative Development.

Merge conflicts have long been recognized as a significant source of coordination overhead in distributed software development [6, 35]. The impact of merge conflicts extends beyond individual repositories. In large software ecosystems, structural interconnectedness means that integration friction in one component can propagate widely [42], reinforcing the need for systematic study of conflict-introducing contributions. Brun et al. [6] demonstrate that collaboration conflicts are frequent and costly, and propose early detection mechanisms to mitigate their impact. Subsequent studies have analyzed the causes and characteristics of merge conflicts in large-scale repositories, highlighting the role of concurrent edits, file centrality, and developer coordination patterns [5, 54]. Research has also investigated conflict prediction and prevention techniques [5]. These approaches leverage historical commit data, code ownership, and file modification patterns to estimate the likelihood of conflicts prior to merging. However, existing conflict datasets primarily focus on human-authored changes and do not explicitly consider contributions generated by AI coding agents, despite recent evidence that AI assistants can increase commit frequency by approximately 13.55% [14].

AgenticFlict extends this line of research by providing a reproducible dataset of textual merge conflicts specifically in the context of Agentic PRS.

Datasets and Benchmarks. Existing merge conflict datasets can be broadly categorized into traditional collaborative benchmarks and emerging AI-centric repositories. Traditional benchmarks focus on human-authored conflicts at scale, such as the 2,731 Java-based projects studied by Ghiotto et al. [18], reporting that nearly 20% of merges require manual intervention and subsequent analyses of conflict structure in 123 Java Projects, revealing that conflicts are primarily concentrated within shared method bodies [1]. More recently, ConflictBench [49] was introduced as a dedicated benchmark specifically designed to evaluate merge tools. It provides a curated collection of conflicting scenarios, categorized by programming language and conflict type. Similarly, datasets like those used

in SBCR [12] focus on the textual similarity between conflict resolutions and their parent versions, offering nearly 10,000 conflict chunks for 1,062 Java projects.

With the rise of Large Language Models (LLMs), AI-centric datasets have emerged to capture interactions between developers and LLMs. DevGPT [58] introduced a dataset of shared ChatGPT conversations linked to GitHub artifacts, later extended by PatchTrack [37, 38] with additional PRs to study the influence of ChatGPT on pull request decision-making and developer-ChatGPT conversation lifecycle. Similarly, AIDev [34] provides a large-scale collection of contributions from multiple AI coding agents. While these datasets offer valuable insights into how AI-generated contributions are created and reviewed, they primarily focus on high-level metadata such as acceptance rates and discussion dynamics. They do not provide explicit or reproducible labels for textual merge conflicts, limiting our ability to systematically quantify the integration friction introduced by AI-generated changes.

6 Threats to Validity

In this section, we discuss potential threats to the validity of the AgenticFlict dataset.

First, conflict detection is based on deterministic local merge simulation using commit identifiers retrieved via the GitHub GraphQL API. In some cases, these references may no longer be available due to repository changes such as force pushes or deletions. We exclude such PRs to avoid incorrect conflict labeling, at the cost of reduced coverage.

Second, we capture merge conflicts using textual conflict markers produced by Git during merge simulation. While this provides a consistent and widely used proxy for integration issues, it does not account for higher-level forms of conflict such as logical inconsistencies or post-merge defects. Furthermore, our analysis focuses on open and closed (unmerged) PRs, which means we may miss conflicts that were previously encountered and resolved during the lifecycle of merged PRs. We adopt this design for a methodological reason: once a PR is merged, Git no longer preserves the pre-merge conflict state. Any conflicts that arose during review were resolved prior to merging, meaning that the conflict markers, line spans, and severity metrics we extract cannot be recovered post-hoc from the merge commit alone. Reconstructing them would require speculative replay of intermediate review states, which we deliberately avoid to preserve reproducibility

Finally, AgenticFlict is constructed on top of the AIDev dataset and therefore inherits its limitations. In particular, the dataset focuses on Agentic PRS and may overrepresent repositories that actively adopt AI tools. As a result, our findings may not generalize to all open-source projects or industrial settings. Extending the dataset to include merge conflicts from human-authored PRs is an important direction for future work. In addition, conflict behavior may vary across programming languages, repository sizes, and development practices.

7 Conclusion

In this paper, we introduced AgenticFlict, a large-scale dataset designed to characterize merge conflicts in AI coding agent PRs. The dataset comprises over 142K Agentic PRs collected from more than 59K repositories, with over 107K successfully analyzed through deterministic merge simulation resulting in over 29K (27.67%) PRs exhibiting textual merge conflicts. Our approach enables reproducible conflict detection and provides fine-grained conflict-region metadata, including conflicting files and line-level spans, resulting in over 336K conflict regions. Our analysis shows that merge conflicts are both frequent and often substantial in AI-generated contributions, highlighting integration as a key challenge in AI-assisted software development. By making these conflict patterns observable at scale, AgenticFlict provides a foundation for studying how AI agents interact with collaborative development workflows. In future work, we plan to extend the dataset to include merge conflicts from human-authored PRs in the same repositories as the Agentic PRs, enabling direct comparative analysis. We plan to investigate conflict characteristics that go beyond textual markers, including logical inconsistencies and post-merge defects, which our current pipeline does not capture. We hope this dataset will support future research on conflict prediction, automated resolution, and the design of tools that better integrate AI-generated code into modern development pipelines. More broadly, our work contributes to understanding the evolving role of AI agents as active participants in Software Engineering 3.0.

Acknowledgments

This research was supported by the National Science Foundation Grant#: 2519136 and the National Science Foundation Major Research Instrumentation (NSF MRI) (Grant#: 2117941).

References

[1] Paola Accioly, Paulo Borba, and Guilherme Cavalcanti. 2018. Understanding semi-structured merge conflict characteristics in open-source Java projects. *Empirical Software Engineering* 23, 4 (2018), 2051–2085. doi:10.1007/s10664-017-9586-1

[2] Adam Alami and Neil Ernst. 2025. Human and Machine: How Software Engineers Perceive and Engage with AI-Assisted Code Reviews Compared to Their Peers. In *2025 IEEE/ACM 18th International Conference on Cooperative and Human Aspects of Software Engineering (CHASE)*. 63–74. doi:10.1109/CHASE66643.2025.00016

[3] Anthropic. 2025. Claude.ai. https://claude.ai/. Accessed: 2025-12-14.

[4] Christian Bird, Peter C. Rigby, Earl T. Barr, David J. Hamilton, Daniel M. German, and Prem Devanbu. 2009. The promises and perils of mining git. In *2009 6th IEEE International Working Conference on Mining Software Repositories*. 1–10. doi:10.1109/MSR.2009.5069475

[5] Caius Brindescu, Iftekhar Ahmed, Rafael Leano, and Anita Sarma. 2020. Planning for untangling: predicting the difficulty of merge conflicts. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering* (Seoul, South Korea) *(ICSE '20)*. Association for Computing Machinery, New York, NY, USA, 801–811. doi:10.1145/3377811.3380344

[6] Yuriy Brun, Reid Holmes, Michael D. Ernst, and David Notkin. 2013. Early Detection of Collaboration Conflicts and Risks. *IEEE Transactions on Software Engineering* 39, 10 (2013), 1358–1375. doi:10.1109/TSE.2013.28

[7] John Businge, Mehrdad Abdi, and Serge Demeyer. 2023. *Analyzing Variant Forks of Software Repositories from Social Coding Platforms*. Springer International Publishing, Cham, 131–152. doi:10.1007/978-3-031-36060-2_6

[8] John Businge, Alexandre Decan, Ahmed Zerouali, Tom Mens, and Serge Demeyer. 2020. An Empirical Investigation of Forks as Variants in the npm Package Distribution. In *Proceedings of the 19th Belgium-Netherlands Software Evolution Workshop, BENEVOL 2020, Luxembourg, December 3-4, 2020 (CEUR Workshop Proceedings, Vol. 2912)*, Mike Papadakis and Maxime Cordy (Eds.). CEUR-WS.org. http://ceur-ws.org/Vol-2912/paper1.pdf

[9] John Businge, Moses Openja, Sarah Nadi, Engineer Bainomugisha, and Thorsten Berger. 2018. Clone-Based Variability Management in the Android Ecosystem. In *International Conference on Software Maintenance and Evolution*. IEEE, 625–634.

[10] John Businge, Moses Openja, Sarah Nadi, and Thorsten Berger. 2022. Reuse and Maintenance Practices among Divergent Forks in Three Software Ecosystems. *Journal of Empirical Software Engineering* 27, 2 (2022), 54. doi:10.1007/s10664-021-10078-2

[11] Heleno de S. Campos Junior, Gleiph Ghiotto L. de Menezes, Márcio de Oliveira Barros, André van der Hoek, and Leonardo Gresta Paulino Murta. 2022. Towards

Merge Conflict Resolution by Combining Existing Lines of Code. In *Proceedings of the XXXVI Brazilian Symposium on Software Engineering* (Virtual Event, Brazil) *(SBES '22)*. Association for Computing Machinery, New York, NY, USA, 425–434. doi:10.1145/3555228.3555229

[12] Heleno de S. Campos Junior, Gleiph Ghiotto L. de Menezes, Márcio de Oliveira Barros, André van der Hoek, and Leonardo Gresta Paulino Murta. 2025. Towards a feasible evaluation function for search-based merge conflict resolution. *ACM Trans. Softw. Eng. Methodol.* (July 2025). doi:10.1145/3748256 Just Accepted.

[13] Valerio Cosentino, Javier L. Cánovas Izquierdo, and Jordi Cabot. 2017. A Systematic Mapping Study of Software Development With GitHub. *IEEE Access* 5 (2017), 7173–7192. doi:10.1109/ACCESS.2017.2682323

[14] Kevin Zheyuan Cui, Mert Demirer, Sonia Jaffe, Leon Musolff, Sida Peng, and Tobias Salz. 2026. The effects of generative AI on high-skilled work: Evidence from three field experiments with software developers. *Management Science* (2026).

[15] Cursor. 2025. Cursor: AI Code Editor. https://cursor.com/. Accessed: 2025-12-14.

[16] Devin AI. 2025. Devin AI — AI Coding Assistant. https://app.devin.ai/. Accessed: 2025-12-14.

[17] Roberto Di Cosmo and Stefano Zacchiroli. 2017. Software heritage: Why and how to preserve software source code. In *iPRES 2017*.

[18] Gleiph Ghiotto, Leonardo Murta, Márcio Barros, and André van der Hoek. 2020. On the Nature of Merge Conflicts: A Study of 2,731 Open Source Java Projects Hosted by GitHub. *IEEE Transactions on Software Engineering* 46, 8 (2020), 892–915. doi:10.1109/TSE.2018.2871083

[19] Git Project. 2026. Partial Clone. https://git-scm.com/docs/partial-clone. Accessed: 2026-04-02.

[20] GitHub Copilot. 2025. GitHub Copilot. https://github.com/copilot. Accessed: 2025-12-14.

[21] GitHub, Inc. 2026. GitHub Terms of Service. https://docs.github.com/en/site-policy/github-terms/github-terms-of-service. Accessed: 2026-04-02.

[22] GitHub, Inc. 2026. GraphQL API. https://docs.github.com/en/graphql. Accessed: 2026-04-02.

[23] GitHub, Inc. 2026. Rate limits and query limits for the GraphQL API. https://docs.github.com/en/graphql/overview/rate-limits-and-query-limits-for-the-graphql-api. Accessed: 2026-04-02.

[24] Ismail Sergen Göçmen, Ahmed Salih Cezayir, and Eray Tüzün. 2025. Enhanced code reviews using pull request based change impact analysis. *Empirical Software Engineering* 30, 3 (2025), 64. doi:10.1007/s10664-024-10600-2

[25] Pavlína Wurzel Gonçalves, Pooja Rani, Margaret-Anne Storey, Diomidis Spinellis, and Alberto Bacchelli. 2025. Code Review Comprehension: Reviewing Strategies Seen Through Code Comprehension Theories. In *2025 IEEE/ACM 33rd International Conference on Program Comprehension (ICPC)*. 589–601. doi:10.1109/ICPC66645.2025.00068

[26] Georgios Gousios. 2013. The GHTorent dataset and tool suite. In *2013 10th Working Conference on Mining Software Repositories (MSR)*. 233–236. doi:10.1109/MSR.2013.6624034

[27] Georgios Gousios, Martin Pinzger, and Arie van Deursen. 2014. An exploratory study of the pull-based software development model. In *Proceedings of the 36th International Conference on Software Engineering* (Hyderabad, India) *(ICSE 2014)*. Association for Computing Machinery, New York, NY, USA, 345–355. doi:10.1145/2568225.2568260

[28] Mário Luís Guimarães and António Rito Silva. 2012. Improving early detection of software merge conflicts. In *2012 34th International Conference on Software Engineering (ICSE)*. 342–352. doi:10.1109/ICSE.2012.6227180

[29] 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:2509.06216 [cs.SE] https://arxiv.org/abs/2509.06216

[30] Ahmed E. Hassan, Gustavo A. Oliva, Dayi Lin, Boyuan Chen, Zhen Ming, and Jiang. 2024. Towards AI-Native Software Engineering (SE 3.0): A Vision and a Challenge Roadmap. arXiv:2410.06107 [cs.SE] https://arxiv.org/abs/2410.06107

[31] Kosei Horikawa, Hao Li, Yutaro Kashiwa, Bram Adams, Hajimu Iida, and Ahmed E Hassan. 2025. Agentic Refactoring: An Empirical Study of AI Coding Agents. *arXiv preprint arXiv:2511.04824* (2025).

[32] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. German, and Daniela Damian. 2014. The promises and perils of mining GitHub. In *Proceedings of the 11th Working Conference on Mining Software Repositories* (Hyderabad, India) *(MSR 2014)*. Association for Computing Machinery, New York, NY, USA, 92–101. doi:10.1145/2597073.2597074

[33] Oleksii Kononenko, Olga Baysal, and Michael W. Godfrey. 2016. Code review quality: how developers see it. In *Proceedings of the 38th International Conference on Software Engineering* (Austin, Texas) *(ICSE '16)*. Association for Computing Machinery, New York, NY, USA, 1028–1038. doi:10.1145/2884781.2884840

[34] 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).

[35] Wardah Mahmood, Moses Chagama, Thorsten Berger, and Regina Hebig. 2020. Causes of merge conflicts: a case study of ElasticSearch. In *Proceedings of the 14th International Working Conference on Variability Modelling of Software-Intensive Systems* (Magdeburg, Germany) *(VaMoS '20)*. Association for Computing Machinery, New York, NY, USA, Article 9, 9 pages. doi:10.1145/3377024.3377047

[36] Shane McKee, Nicholas Nelson, Anita Sarma, and Danny Dig. 2017. Software Practitioner Perspectives on Merge Conflicts and Resolutions. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 467–478. doi:10.1109/ICSME.2017.53

[37] Daniel Ogenrwot and John Businge. 2024. PatchTrack: Analyzing ChatGPT's Impact on Software Patch Decision-Making in Pull Requests. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering* (Sacramento, CA, USA) *(ASE '24)*. Association for Computing Machinery, New York, NY, USA, 2480–2481. doi:10.1145/3691620.3695338

[38] Daniel Ogenrwot and John Businge. 2025. PatchTrack: A Comprehensive Analysis of ChatGPT's Influence on Pull Request Outcomes. arXiv:2505.07700 [cs.SE] https://arxiv.org/abs/2505.07700

[39] Daniel Ogenrwot and John Businge. 2025. Refactoring-Aware Patch Integration Across Structurally Divergent Java Forks. In *2025 IEEE International Conference on Source Code Analysis & Manipulation (SCAM)*. 25–36. doi:10.1109/SCAM67354.2025.00010

[40] Daniel Ogenrwot and John Businge. 2026. *AgenticFlict: A Large-Scale Dataset of Merge Conflicts in AI Coding Agent Pull Requests on GitHub*. doi:10.5281/zenodo.19396916

[41] Daniel Ogenrwot and John Businge. 2026. How AI Coding Agents Modify Code: A Large-Scale Study of GitHub Pull Requests. arXiv:2601.17581 [cs.SE] https://arxiv.org/abs/2601.17581

[42] Daniel Ogenrwot, John Businge, and Shaikh Arifuzzaman. 2026. Structural and Connectivity Patterns in the Maven Central Software Dependency Network. In *Software and Data Engineering*, Nick Rahimi, Venkat Margapuri, and Noor Amiri Golilarz (Eds.). Springer Nature Switzerland, Cham, 129–151.

[43] OpenAI. 2025. Codex — OpenAI. https://openai.com/codex/. Accessed: 2025-12-14.

[44] Moein Owhadi-Kareshk, Sarah Nadi, and Julia Rubin. 2019. Predicting Merge Conflicts in Collaborative Software Development. In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 1–11. doi:10.1109/ESEM.2019.8870173

[45] Sida Peng, Eirini Kalliamvakou, Peter Cihon, and Mert Demirer. 2023. The Impact of AI on Developer Productivity: Evidence from GitHub Copilot. *arXiv preprint arXiv:2302.06590* (2023). doi:10.48550/arXiv.2302.06590 Submitted on 13 Feb 2023.

[46] Poedjadevie Kadjel Ramkisoen, John Businge, Brent van Bladel, Alexandre Decan, Serge Demeyer, Coen De Roover, and Foutse Khomh. 2022. PaReco: patched clones and missed patches among the divergent variants of a software family. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Singapore, Singapore) *(ESEC/FSE 2022)*. Association for Computing Machinery, New York, NY, USA, 646–658. doi:10.1145/3540250.3549112

[47] Henrique Rocha and John Businge. 2022. Blockchain-Oriented Software Variant Forks: A Preliminary Study. In *5th International Workshop on Blockchain Oriented Software Engineering*.

[48] Bowen Shen, Muhammad Ali Gulzar, Fei He, and Na Meng. 2023. A Characterization Study of Merge Conflicts in Java Projects. *ACM Trans. Softw. Eng. Methodol.* 32, 2, Article 40 (March 2023), 28 pages. doi:10.1145/3546944

[49] Bowen Shen and Na Meng. 2024. ConflictBench: A benchmark to evaluate software merge tools. *Journal of Systems and Software* 214 (2024), 112084. doi:10.1016/j.jss.2024.112084

[50] Jeffrey Svajlenko, Judith F. Islam, Iman Keivanloo, Chanchal K. Roy, and Mohammad Mamun Mia. 2014. Towards a Big Data Curated Benchmark of Inter-project Code Clones. In *2014 IEEE International Conference on Software Maintenance and Evolution*. 476–480. doi:10.1109/ICSME.2014.77

[51] Alexey Svyatkovskiy, Sarah Fakhoury, Negar Ghorbani, Todd Mytkowicz, Elizabeth Dinella, Christian Bird, Jinu Jang, Neel Sundaresan, and Shuvendu K. Lahiri. 2022. Program merge conflict resolution via neural transformers. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Singapore, Singapore) *(ESEC/FSE 2022)*. Association for Computing Machinery, New York, NY, USA, 822–833. doi:10.1145/3540250.3549163

[52] Priyan Vaithilingam, Zheng Xu, and Elena L. Glassman. 2023. Copilot or Co-Author? Examining the Role of Code Generation Tools in Collaborative Programming. In *Proceedings of the 2023 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, USA.

[53] Priyan Vaithilingam, Tianyi Zhang, and Elena L. Glassman. 2022. Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. ACM, USA, 1–17.

[54] Gustavo Vale, Eduardo Fernandes, Eduardo Figueiredo, and Sven Apel. 2023. Behind Developer Contributions on Conflicting Merge Scenarios. In *2023 IEEE 23rd International Working Conference on Source Code Analysis and Manipulation (SCAM)*. 25–36. doi:10.1109/SCAM59687.2023.00014

[55] Gustavo Vale, Claus Hunsen, Eduardo Figueiredo, and Sven Apel. 2022. Challenges of Resolving Merge Conflicts: A Mining and Survey Study. *IEEE Transactions on Software Engineering* 48, 12 (2022), 4964–4985. doi:10.1109/TSE.2021.3130098

[56] Manushree Vijayvergiya, Małgorzata Salawa, Ivan Budiselić, Dan Zheng, Pascal Lamblin, Marko Ivanković, Juanjo Carin, Mateusz Lewko, Jovan Andonov, Goran Petrović, Daniel Tarlow, Petros Maniatis, and René Just. 2024. AI-Assisted Assessment of Coding Practices in Modern Code Review. In *Proceedings of the 1st ACM International Conference on AI-Powered Software* (Porto de Galinhas, Brazil) *(AIware 2024)*. Association for Computing Machinery, New York, NY, USA, 85–93. doi:10.1145/3664646.3665664

[57] 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:2509.14745 [cs.SE] https://arxiv.org/abs/2509.14745

[58] Tao Xiao, Christoph Treude, Hideaki Hata, and Kenichi Matsumoto. 2024. DevGPT: Studying Developer-ChatGPT Conversations. In *Proceedings of the 21st International Conference on Mining Software Repositories* (Lisbon, Portugal) *(MSR '24)*. Association for Computing Machinery, New York, NY, USA, 227–230. doi:10.1145/3643991.3648400

[59] Albert Ziegler, Eirini Kalliamvakou, X. Alice Li, Andrew Rice, Devon Rifkin, Shawn Simister, Ganesh Sittampalam, and Edward Aftandilian. 2022. Productivity assessment of neural code completion. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming* (San Diego, CA, USA) *(MAPS 2022)*. Association for Computing Machinery, New York, NY, USA, 21–29. doi:10.1145/3520312.3534864

Received 2026-02-15; accepted 2026-03-28