How AI Coding Agents Resolve Merge Conflicts: An Empirical Study

Heleno de Souza Campos Junior
Federal Fluminense University
Niterói, Brazil
Federal University of Rio de Janeiro
Rio de Janeiro, Brazil
heleno@cos.ufrj.br

Leonardo Gresta Paulino Murta
Federal Fluminense University
Niterói, Brazil
leomurta@ic.uff.br

Abstract

AI coding agents such as Claude Code, OpenAI Codex, GitHub Copilot, Cursor, and Devin now open pull requests (PRs) at scale, making merge conflict resolution an increasingly important part of AI-assisted software development. Yet no prior study has examined who resolves these conflicts or how conflict resolution strategies differ across agents and from humans. We present the first study of who resolves these conflicts and how conflict resolution strategies differ across agents and from humans, analyzing 14,960 conflicting *internal merge commits* from 12,299 AI-authored PRs. Three findings emerge. First, agents rarely resolve the conflicts they produce: humans author 14,386 (96.1%) of conflicting merge commits, and agents' *self resolution rates* differ by nearly 60× across vendors. Second, agents exhibit no common conflict resolution behaviour. Strategy profiles range from statistically indistinguishable from humans to strongly divergent, while pairwise agent comparisons range from near identical to almost disjoint. Third, two of the five agents derive more than 80% of their observed *self resolution profile* from a single repository, revealing a *single repository concentration* that must be accounted for before interpreting per agent behaviour. These findings argue against treating AI coding agents as a homogeneous class and instead motivate vendor specific merge tooling, evaluation, and CI safeguards.

Keywords

Merge Conflicts, AI Agents, Conflict Resolution, Software Merging

1 Introduction

The increasing adoption of AI-powered coding agents, such as Claude Code, OpenAI Codex, GitHub Copilot, Cursor, and Devin, represents a paradigm shift in software development workflows. These agents autonomously create branches, implement features, and open pull requests (PRs) in open-source repositories, contributing code alongside human developers at a scale that only became possible in the last two years. The AIDev catalogue [28] alone documents over 932,000 AI-authored PRs across 116,000 repositories, positioning agentic contributions as a first-class citizen of modern collaborative development.

When an AI-authored branch needs to incorporate upstream changes (e.g., via `git merge main`), textual merge conflicts may arise. These conflicts must be resolved before the merged code can be committed, and the resolution may be performed by the AI agent itself, by a human developer, or collaboratively. Despite the scale of agentic contributions, two questions remain unanswered for this new population of merges: *who* performs the integration step on AI-authored branches, and *how* the agents that do their own merges behave relative to each other and to humans.

Prior work on merge conflict resolution has focused exclusively on human developers [18, 19, 22, 37, 38]; the one dataset that targets agentic PRs, AgenticFlict [31], analyzes merges between the PR head and base branches, but does not distinguish AI from human actors in the integration step. In addition, they do not study the resolution strategies used for resolving these conflicts. Consequently, no prior study has separated AI-performed from human-performed resolutions on merge commits and characterized the strategies applied by each agent individually.

In this paper, we address these gaps by conducting an empirical study on internal merge commits within AI-authored pull requests from the AIDev catalogue [28]. We identify and reproduce each internal merge, localize the developer's resolution for every conflicting chunk following Dinella et al. [18] strategy, classify each resolution using the Ghiotto et al. [22] taxonomy, and attribute each resolution to an AI agent or a human based on the committer identity [27]. Our main contributions are: (i) the first attribution of conflict resolutions to specific AI coding agents on real merge commits at scale; (ii) a characterization of the resolution strategies used by AI agents and humans; (iii) a replication package containing all collection and analysis scripts. Concretely, we answer the following research questions:

- **RQ1: Who resolves the merge conflicts introduced in AI-authored PRs, and how does this vary across coding agents?** Whether conflicts are resolved by the agent autonomously or by a human has direct implications for pipeline design and trust in agentic workflows. We identify who commits each internal merge and how the propensity for self-resolution differs by agent.
- **RQ2: How do AI coding agents resolve their own merge conflicts, both individually and relative to one another?** Aggregating across agents can mask large vendor-specific differences. We compare each agent's per-agent resolution-strategy distribution against the human reference within our corpus, run pairwise agent-against-agent contrasts, stratify the heterogeneity by file category, and surface the repository-concentration that confounds two of the five agent profiles.

Our headline results, developed in Sections 5.2–5.3, are: among the 14,960 conflicting internal merges, humans commit 96.1% and the per-agent self-resolution rate spans nearly 60× (Codex 0.5% to Devin 29.4%); the five agents differ markedly from each other in resolution strategy (ranging from statistically indistinguishable

from humans to strongly divergent, with inter-agent comparisons spanning from near equivalence to almost completely disjoint behaviour); and two of the five agents (Codex and Claude Code) are each dominated by a single repository (> 83% of their self-resolved conflicting merges in each case), so their per-agent profile reflects the dominant repository's workflow, not the agent.

The remainder of the paper is organized as follows. Section 2 presents background, Section 3 discusses related work, Section 4 describes our methodology and pipeline, Section 5 reports results for each RQ, Section 6 discusses implications, Section 7 addresses threats to validity, and Section 8 concludes.

2 Background

This section introduces the mechanics of merge conflicts and the human-centric taxonomy of resolution strategies that our study reuses and extends to AI coding agents.

2.1 Software Merging and Merge Conflicts

Collaborative development relies on branching, whereby developers work in parallel and later reconcile their contributions via *merging* [29]. Textual three-way merge, as implemented in tools such as `git merge`, automatically combines non-overlapping edits and reports a *conflict* when changes overlap [14, 22, 29]. The `diff3` output additionally preserves the common ancestor between `|||||||` and `=======`, providing a three-way view on which subsequent analyses can rely [18, 19].

Merge conflicts are frequent and time-consuming. Ghiotto et al. [22] inspected 960,366 merge scenarios across 2,731 Java projects and found that 11.9% resulted in at least one conflict; comparable rates (10–20%) have been reported in other ecosystems [26, 38]. Vale et al. [38] surveyed developers and identified four recurring challenges: understanding the conflict, deciding on a resolution, dealing with tooling, and coordinating with teammates. Nelson et al. [30] complemented these findings with a life-cycle perspective, characterizing the understand–plan–resolve–verify workflow. Beyond line-based merging, semistructured approaches [1, 5, 14] reduce spurious conflicts, while proactive approaches [9, 25] warn developers in advance. Finally, Boll et al. [7] showed that 87.9% of conflicting chunks follow a small, language-agnostic catalogue, suggesting that semi-automation may be more feasible than previously assumed.

2.2 Conflict Resolution Taxonomy

Ghiotto et al. [22] introduced the canonical taxonomy that classifies each resolved chunk by comparing the merged text against the two conflicting versions (v_1, v_2): the developer may adopt one version verbatim (*V1*, *V2*), concatenate both sides in either order (*Concatenation*), interleave lines from both (*Combination*), write content absent from both sides (*New Code*), or resolve to an empty region (*None*). Across the 175,805 Java chunks they studied, *V1* accounts for 50% of the resolutions, *V2* for 25%, *New Code* for 13%, *Combination* for 9%, and *Concatenation* for 3%. A very small percentage (0.07%) of the resolutions were classified as *None*. We adopt this taxonomy, which gives us six classifiable labels (V1, V2, CC, CB, NC, NN), directly comparable to the human baseline.

3 Related Work

We now turn to the two bodies of work our study bridges: automated approaches to merge conflict resolution, and empirical characterizations of AI coding agents as software contributors.

3.1 Automated Merge Conflict Resolution

ML-based resolvers have matured along two paradigms, both focused on *how* a conflict should be resolved rather than *who* authored the conflicting code. This is the distinction our own attribution-focused study is built around. The *classification* paradigm (e.g., DeepMerge [18] and MergeBERT [37]) trains models to select a strategy label. The *generation* paradigm (MergeGen [19], ChatMerge [36]) synthesizes resolution text directly, capturing *New Code* resolutions that label-pickers cannot emit. Search-based approaches (SBCR [10], Campos Junior et al. [11]) recommend resolutions using lines already present in the conflicting versions, and further specialized methods include template-based synthesis [33] and data-mining recommenders (MESTRE [21], RPredictor [3]). A related line of work targets conflict *prediction* rather than resolution, warning developers before a conflict occurs [8, 32, 35]; since prediction does not itself resolve a conflict, we do not discuss it further here. These resolvers are natural complements to our findings: as we show in Section 5.2, humans currently resolve the vast majority of agent-authored conflicts, so an ML-based resolver is a plausible candidate to absorb part of that remaining burden, particularly for the short, low-complexity conflicts that dominate our corpus (Section 5.1).

Research Gap. All the studies above treat *humans* as the implicit resolvers. No prior work has separated AI-performed from human-performed conflict resolutions on AI-authored branches, characterized how individual coding agents differ in their resolution strategies, or examined how robust an aggregate agent-versus-human contrast is to the per-agent or per-repository composition of the corpus. Our work fills these gaps by reusing Ghiotto et al. [22] taxonomy on a population of merges that only became possible after the recent rise of autonomous coding agents and by reporting per-agent contrasts and explicit robustness checks alongside the aggregate.

3.2 AI Agents in Software Development

The past two years have seen the emergence of autonomous coding agents that act as first-class contributors on GitHub. Li et al. [27] label this regime *Software Engineering 3.0*, in which agents open branches, write code, run tests, and submit pull requests reviewed by humans or other agents. Peng et al. [34] reported a 55% speed-up in a randomized controlled trial of GitHub Copilot, and Wang et al. [39] survey LLM-based agent architectures and benchmarks, prominently SWE-bench [24].

Empirical studies of these agents remain scarce. Li et al. [27, 28] characterize agentic PRs along dimensions such as acceptance rate, size, and review turnaround, but do not examine how branches are integrated. Ogenrwot and Businge [31] study simulated merges between the PR head and base commits to count conflicts, but do not examine the resolutions adopted by the developers (human or AI). Earlier studies of bot-authored PRs [4] established the methodology for studying bot contributions, but modern agents synthesize

open-ended code rather than applying fixed templates, warranting dedicated study. We complement this prior work by analyzing merge commits within PR branches, distinguishing AI- from human-driven resolutions, and applying the human-centric taxonomy at the per-agent level rather than only on the aggregate.

3.3 The AIDev and AgenticFlict Datasets

Two datasets currently underpin empirical research on AI coding agents, but neither captures how conflicts are actually resolved. **AIDev** [28], the official dataset for the MSR Challenge 2026, catalogues 932,791 PRs opened by five agents (Claude Code, Codex, Copilot, Cursor, Devin) across 116,211 repositories and 72,631 developers. The data are mined from the GitHub Archive and include rich PR-level metadata (title, body, diff summary, state, `base_oid`, `head_oid`, `merged_at`), but no information about merge conflicts or their resolution. While prior work has extensively analyzed AIDev, we are not aware of studies that analyze merge resolution behaviour within AI-authored PRs or characterize agent-specific strategies.

AgenticFlict [31] extends AIDev by locally cloning each repository and running a `git merge` between the PR's `base_oid` and `head_oid`, identifying 29,609 conflicting PRs (27.67% rate) and 336,380 conflict regions. This shows that conflicts in agent-authored code are frequent and non-trivial, but it has two limitations that motivate our work: (i) the conflicts are *simulated* from the PR's tip commits and sometimes may not correspond to a real merge ever committed, so there are no *resolutions* to study; and (ii) no resolver actor exists in the simulated setup, so AI and human cannot be distinguished in integrations. Our study addresses both gaps from AIDev directly: we look *inside* each PR branch for real two-parent commits ("internal merges"), reproduce them with `git merge` and `diff3` output, localize the accepted resolution for every conflicting chunk, and attribute each resolution to the merge committer.

4 Methodology

This study employs a five-stage pipeline, illustrated in Figure 1, that progresses from raw pull request data to the classification and analysis of merge conflict resolutions. We begin with the AIDev dataset, filter and reconstruct commit histories for in-scope pull requests, extract and deduplicate internal merge commits, and reproduce these merges to obtain fine-grained conflict chunks. These reconstructed conflicts are then localized within the resolved code and analyzed through strategy classification and resolver attribution. Each stage of this pipeline corresponds to one component of Figure 1 and is described in detail below.

4.1 Data Source

Our study is scoped to the full AIDev catalogue [28] (approximately 932,000 PRs from five agents across over 116,000 repositories). We consider PRs in all states (*merged*, *closed*, *open*); restricting to merged PRs would bias toward histories rewritten by rebase or squash-merge and drop the intermediate merge commits we want to study.

We target the same five representative agents used in AIDev: Anthropic's **Claude Code**, OpenAI's **Codex**, GitHub's **Copilot** (agentic mode), **Cursor**, and Cognition's **Devin**. Each authors commits under a distinctive bot identity (e.g., `claude[bot]`, `copilot[bot]`, `devin-ai-integration[bot]`), which makes attribution possible at scale.

4.2 PR Filtering and Commit Retrieval

The per-PR commit data needed to locate internal merges is not shipped with AIDev, so we produce it ourselves by fetching `refs/pull/*/head` for every in-scope repository and enumerating, per PR, the commits between the fork-point with the base branch and the PR tip.

Applying this procedure yields a universe of 210,902 in-scope PRs across 57,582 repositories; the complete SHA list could not be found for 25,571 of these PRs (12.1%), mostly because of disconnected histories (20,439) or PR head references no longer reachable (5,132), leaving 185,331 PRs with a fully recoverable commit history for the merge-extraction stage that follows.

4.3 Merge Extraction and Deduplication

For each PR, we identify merge commits (commits with exactly two parents). These are merges created to bring upstream changes into the feature branch, distinct from the single "integration merge" that GitHub creates when a PR is accepted. We exclude octopus merges (three or more parents), following Ghiotto et al. [22], and the GitHub integration merge itself: any two-parent commit whose message matches `^Merge pull request #d+` (case-insensitive) is filtered out, since GitHub's pre-merge already check them for conflicts before integration. This step excluded 70,042 integration-merge SHAs and 93 octopus-merge SHAs.

Merge-level deduplication. In the commit graph of Git, a merge commit is a first-class object that can be reachable from multiple branch tips. Consequently, the same merge commit may appear in the head histories of several PRs on GitHub (e.g., due to force-pushes, PR re-openings, branch reuse, or cross-branch merges), causing naive PR-scoped indexing to overcount the same artefact by attributing it to each PR in whose history it is reachable. To eliminate this duplication, we define merge- and chunk-level statistics using PR-independent natural keys: (repository, merge SHA) for merges and (repository, merge SHA, file path, chunk index) for chunks, so that each merge commit is counted exactly once regardless of how many PRs reference it.

Applying this deduplication reduces 452,114 raw PR-level merge references to 50,700 distinct merge commits; in the most extreme case, a single merge commit is reachable from 848 distinct PRs. Overall, only 12,299 PRs (5.8% of the in-scope PRs) contain at least one internal merge.

4.4 Conflict Reconstruction

Conflict extraction via three-way merge. For each merge commit with parents p_1 and p_2, we compute the merge base (b = merge-base(p_1, p_2)) and identify files modified on both sides using `git merge-tree`. For each such file, we reproduce the three-way merge using:

```
git merge-file -p --diff3 p1_file base_file p2_file
```

We parse the resulting diff3 output to extract individual conflict chunks, each delimited by `<<<<<<<`, `|||||||` , `=======`, and `>>>>>>>` markers. Each chunk produces three text regions: *v1* (from

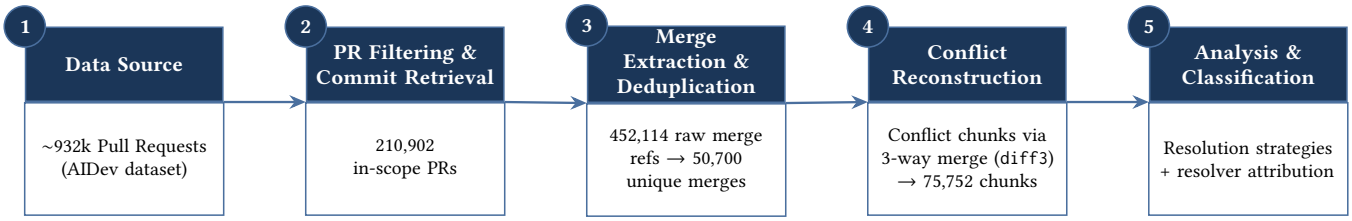

Figure 1: Overview of the data collection and analysis pipeline.

p_1), *base* (from b), and *v2* (from p_2). For each chunk we also record the non-blank line count of each side, computed after trimming whitespace, consistent with Ghiotto et al. [22] normalization.

Resolution localization. For each conflict chunk, we retrieve the resolved file from the merge commit and apply the LocalizeResRegion algorithm of Dinella et al. [18], which isolates the specific resolution text by anchoring on the unique textual context (prefix and suffix) surrounding the conflict region in the three-way merge output. If unique anchors cannot be found, the chunk is marked *Imprecise*. Dinella et al. [18] validated the algorithm on 174 conflict regions from 50 randomly selected merges and found 100% correct localization (95% CI: [0.979, 1.0]); we use it on this basis. We also record the non-blank line count of the resolution text.

A potential confound is a *differential Imprecise rate* across resolver types: if localization fails at different rates for agent- and human-resolved chunks, the observed strategy distributions may not be directly comparable. To bound this effect, we conduct a sensitivity analysis that reassigns all Imprecise chunks to V_1 (upper bound for V_1 dominance) or to NC (lower bound), and assess whether the per-agent contrasts survive across both extremes. The results of this analysis are reported in Section 7.

4.5 Analysis and Classification

Resolution strategy classification. We classify each resolved chunk into one of six resolution strategies, following the taxonomy of Ghiotto et al. [22], by comparing the normalized resolution text against both conflict sides:

- **V1** / **V2**: The resolution is identical to one side of the conflict.
- **CC** (*Concatenation*): The resolution is the concatenation of both sides in either order (we fold both orderings into *CC* to stay comparable with Ghiotto et al. [22]).
- **CB** (*Combination*): The resolution interleaves lines from both sides without introducing new lines.
- **NC** (*New Code*): The resolution introduces content not present in either side.
- **NN** (*None*): The resolution is empty.
- **Imprecise**: The resolution could not be reliably isolated (Section 4.4) and is kept as a separate bucket to avoid silent misclassification.

Because Ghiotto et al. [22] do not report an *Imprecise* bucket, RQ2 distributions are reported over the six classifiable labels, with the *Imprecise* share stated separately.

File-Path categorisation. To condition the strategy analysis on what kind of file each conflict occurs in, we assign each conflicting file to one of six mutually exclusive categories using a deterministic, priority-ordered heuristic applied to the repository-relative file path. The categories and their matching rules are as follows, applied in the order listed (the first matching rule wins):

- **lock**: the file's basename exactly matches one of 16 known dependency lock-file names across all major ecosystems represented in the dataset: `package-lock.json`, `yarn.lock`, `pnpm-lock.yaml`, `npm-shrinkwrap.json`, `Pipfile.lock`, `poetry.lock`, `uv.lock`, `Gemfile.lock`, `composer.lock`, `Cargo.lock`, `go.sum`, `mix.lock`, `flake.lock`, `pubspec.lock`, `Podfile.lock`, and `bun.lockb`.
- **generated**: the file path ends with one of the suffixes `.min.js`, `.min.css`, `.bundle.js`, `.map`, or `.snap`; or the path traverses a known build-output or dependency directory (`dist/`, `build/`, `out/`, `node_modules/`, `vendor/`, `target/`, `coverage/`, `__pycache__/`, `.next/`, `.nuxt/`, `.cache/`, or `.parcel-cache/`).
- **doc**: the file extension is one of `.md`, `.rst`, `.txt`, or `.adoc`.
- **config**: the file extension is one of `.json`, `.yaml`, `.yml`, `.toml`, `.ini`, `.cfg`, `.conf`, or `.env`.
- **migration**: the file extension is `.sql` or the path contains a `/migrations/` or `/migration/` segment.
- **source**: all remaining files, treated as source code.

The heuristic uses both the file name and path. Its known limitation, that some files (e.g., `package.json` in a monorepo) can be functionally generated artefacts yet fall into *config* rather than *generated*, is discussed in Section 7.

Resolver attribution. We determine who resolved each conflict from the author metadata of the merge commit, using a substring/-suffix heuristic over the author login:

- **Agent**: the login ends in `[bot]`, or contains any of `copilot`, `devin`, `claude`, `cursor`, `openai`.
- **Agent-assisted**: the login does not match *Agent*, but a `Co-authored-by:` trailer in the commit message does.
- **Human**: neither the login nor a co-author trailer matches.

The PR-authoring agent is recorded separately, taken from AIDev's bot-account mapping [27, 28], and is distinct from this resolver label. When per-agent strategy distributions are reported (Section 5.3.1 onward), we restrict to chunks where the resolver is *Agent* and proxy the specific agent identity with the PR-authoring agent (i.e., we assign the resolving agent to whichever agent authored the PR). This proxy can misattribute a chunk if a different agent commits inside an already-open PR; the known failure modes are discussed in Section 7.

Analyses per research question. For **RQ1**, we report the distribution of resolver type (agent-resolved, agent-assisted, human-resolved) at the merge-commit level, globally and per agent. For **RQ2**, we report the relative frequency of each resolution label (V1, V2, CC, CB, NC, NN) at the chunk level for each agent and for the human-resolved subset, with the *Imprecise* share reported separately. We then test the heterogeneity of the strategy distribution across agents by (i) comparing each agent against the human-resolved subset of our own corpus, (ii) running pairwise agent-against-agent contrasts with Holm-corrected p-values across the ten agent pairs, and (iii) stratifying each agent's strategy distribution by conflicting file category (*source*, *lock*, *config*, *generated*, *doc*, *migration*). We complement these contrasts with a per-agent repository-concentration analysis (top-1 repo share among self-resolved merges) and a focused case study of the two most concentrated agents.

Baseline calibration. The strategy distribution of the 65,798 chunks resolved by humans in our own corpus matches the Java baseline of Ghiotto et al. [22] to within five percentage points on every strategy (V_1: 45.1 vs. 50.0; V_2: 26.0 vs. 25.0; CC: 6.8 vs. 3.0; CB: 9.9 vs. 9.0; NC: 11.4 vs. 13.0; NN: 0.7 vs. 0.0). On this basis, the human-resolved subset of our corpus is used throughout the paper as the single human reference distribution, as it is in the same context of the agent-resolved and agent-assisted subsets; comparisons against Ghiotto et al. [22] are not revisited in the results.

Statistical analyses. All proportions are reported with 95% Wilson confidence intervals [40]. For association between categorical variables (RQ1, RQ2) we use chi-squared tests with the bias-corrected Cramér's V [6] as primary effect-size metric. Cramér's V ranges from 0 (independence between the two distributions) to 1 (perfect association), with conventional interpretation thresholds of V <0.10 (weak), 0.10–0.30 (moderate), and ≥0.30 (strong). At n≈60,000, even small deviations from independence yield extremely small p-values, making statistical significance less informative than effect size [2, 17], so magnitude carries more information than significance. Conflict chunks within the same merge and repository are not fully independent, so p-values are anti-conservative; effect sizes (V) should therefore be interpreted cautiously as potentially inflated. For per-merge structural distributions (e.g., chunks per merge) we use Kruskal–Wallis with post-hoc Dunn tests [20] with Holm-adjusted p-values [23]; effect sizes are Cliff's δ [15] since distributions are right-skewed. Cramér's V is computed as a descriptive measure of association regardless of sample size, but the chi-squared p-value is reported only when its distributional assumptions hold (rule of thumb: no more than 20% of category combinations have expected counts below five [16]). When this criterion is not met, V is still reported but no p-value is given and the pair is excluded from the Holm correction procedure.

5 Results

This section presents the findings of our study, beginning with a high-level characterization of the dataset (Section 5.1) that establishes the frequency and structural properties of internal merge conflicts. We then address **RQ1** by quantifying the extent to which agents autonomously resolve these conflicts versus relying on human intervention (Section 5.2). Finally, in **RQ2**, we analyze the specific resolution strategies employed by agents, examining how their behaviors diverge from human baselines and investigating the influence of repository-specific patterns on these outcomes (Section 5.3).

Table 1: Dataset overview (after merge-level deduplication).

Metric	Count	Share
Repositories	4,806	—
PRs with ≥ 1 internal merge	12,299	—
PRs with ≥ 1 conflicting merge	7,268	59.1% of PRs
Internal merge commits (dedup.)	50,700	—
Conflicting merge commits	14,960	29.5% of merges
Conflict chunks	121,599	—
Chunks with localized resolution	83,043	68.3% of chunks
of which: Postponed	7,291	6.0% of chunks
Resolved chunks	75,752	62.3% of chunks

5.1 Dataset Overview

After merge-level deduplication, our pipeline covers 50,700 distinct internal merge commits authored inside the 12,299 PRs across 4,806 repositories (Table 1). Of those merges, 14,960 (29.5%) produce at least one textual conflict, yielding 121,599 conflict chunks. At the PR level, 12,299 PRs (5.8% of all in-scope PRs) contribute at least one internal merge and 7,268 (3.4%) contain at least one conflicting merge, indicating that internal merges are the exception rather than the rule in agent-authored PRs and that conflicts, when they do arise, cluster inside a comparatively small slice of the dataset. The resolution-localization algorithm of Dinella et al. [18] uniquely identifies the resolution region for 83,043 of the 121,599 chunks (68.3%); the remaining 38,556 (31.7%) failed to find unique anchors and are kept as *Imprecise*. Within the localized chunks, 7,291 (6.0% of all chunks) are *Postponed*: localization succeeded but the committed file still contained unresolved conflict markers (<<<<<<<, =======, >>>>>>>), meaning the conflict was committed without being resolved. We fold Postponed into the analytical Imprecise bucket alongside anchor failures (combined 45,847 chunks, 37.7%) to avoid silent misclassification; the Postponed sub-category is analyzed further in Section 5.3.3. The remaining 75,752 chunks, for which a resolution could be found, constitute the dataset used in all subsequent analyses.

Structural shape of the conflict data. For context, conflict size in internal merges from agent-authored PRs is heavily right-skewed (Figure 2). Across the 14,960 conflicting merges, the median number of chunks per merge is 2, the 75th percentile is 4, the 90th is 10, but the mean is 8.13 because the tail reaches 9,791 chunks in a single merge; excluding that one merge, the mean drops to 7.47 chunks per merge (111,808 chunks across the remaining 14,959 merges), so the reported mean is not driven by a broad heavy tail but primarily by this single outlier. Per-chunk side sizes are similarly small (median v_1=3 LOC, v_2=2 LOC, resolution 1 LOC), with a heavy tail (99th percentile 275, 150, 1,407 LOC respectively). The merge-level conflict incidence, i.e., the fraction of internal merges that produce at least one textual conflict, ranges from 10.5% (PRs

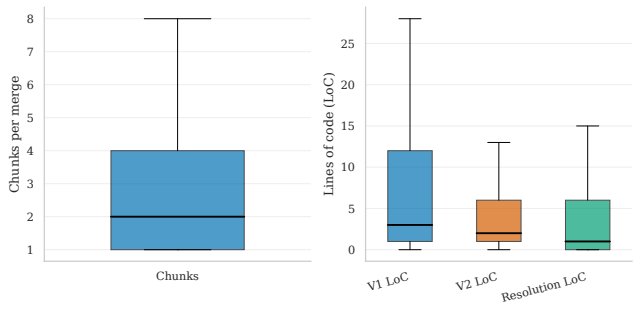

Figure 2: Size distributions in internal merges from agent-authored PRs (outliers omitted for clarity). *Left:* number of conflicting chunks per merge commit (conflicting merges only, n=14,960). *Right:* non-blank LOC for the v_1, v_2, and resolution sides of the 75,752 localized and resolved conflict chunks.

Table 2: Scope and conflict incidence by PR-authoring agent. *Rate* is the fraction of internal merges that produce at least one textual conflict, computed as Conf. merges / Int. merges per agent.

Agent	PRs	Int. merges	Conf. merges	Rate
OpenAI Codex	8,984	29,250	11,684	39.9%
Copilot	1,670	17,216	1,809	10.5%
Devin	814	2,369	763	32.2%
Cursor	551	1,013	484	47.8%
Claude Code	280	852	220	25.8%

authored by Copilot) to 47.8% (PRs authored by Cursor), a more than 4× spread (Table 2). A Kruskal–Wallis test on the number of conflicting chunks per conflicting merge across the different PR-authoring agents yields H=92.1 (p<0.001), with all post-hoc Dunn pairwise effects negligible to small (Cliff's δ<0.17); structural differences across the different PR-authoring agents are detectable but not practically large, and serve mainly as context for the strategy heterogeneity examined in RQ2.

5.2 RQ1: Who Resolves the Conflicts?

Overall resolver distribution. Although we observed 14,960 conflicting internal merges in agent-authored PRs, most of these were actually merged by *humans*: 14,381 (96.1%). Just 462 (3.1%) were merged by the *agent* and 117 (0.8%) are *agent-assisted*, i.e., merges where an agent-authored trailer (e.g., `Co-authored-by: devin-ai-integration[bot]`) appears alongside a human committer. Even when an internal merge within an agent-authored PR *produces a conflict*, requiring an explicit resolution, the integration step remains overwhelmingly human-led: humans perform the merge in about 96% of the cases.

Per-agent contingency. Table 3 cross-tabulates the PR-authoring agent against the resolver of internal merge conflicts. We observe substantial variation in autonomy across agents: Devin exhibits the highest self-resolution rate (29.4%, 224/763), followed by Cursor (8.9%, 43/484), Copilot (6.7%, 122/1,809), Claude Code (5.5%, 12/220), and OpenAI Codex (0.5%, 61/11,684). Claude Code stands out in terms of interaction mode, being the only agent with a sizable assisted share (24.1%, 53/220); whereas assisted resolutions remain below 1% for all others. When considering the composition of the agent-resolved pool (n = 462), the distribution is highly concentrated: Devin and Copilot together account for 74.9% of all agent-resolved conflicting merges (224 and 122 cases, respectively). Notably, this dominance occurs despite OpenAI Codex contributing the largest volume of total merges, indicating that high usage does not translate into autonomous conflict resolution.

Table 3: Resolver of the conflicting internal merges by PR-authoring agent.

PR author	# Human	# Assisted	# Agent	# Total
Devin	537	2	224	763
Cursor	437	4	43	484
Copilot	1,684	3	122	1809
Claude Code	155	53	12	220
OpenAI Codex	11,568	55	61	11,684
Total	14,381	117	462	14,960

Statistical tests. A chi-squared test of independence between the resolver type (agent / human / agent-assisted) and the PR-authoring agent over the 14,960 conflicting merges yields $\chi^2(8)$=3,724.5, p<0.001, Cramér's V=0.35 (strong effect), confirming sharp heterogeneity in resolver propensity across the five agents. Per-agent Wilson 95% CIs on the self-resolution rate clearly separate Devin (29.4%, [26.2%, 32.7%]) from Cursor (8.9%, [6.7%, 11.8%]), Copilot (6.7%, [5.7%, 8.0%]), Claude Code (5.5%, [3.1%, 9.3%]), and Codex (0.5%, [0.4%, 0.7%]). The agent-assisted bucket is dominated by Claude Code (24.1%, [18.9%, 30.2%]); for the other four agents the assisted CI is below 1%.

Finding 1. Conflict resolution in agent-authored PRs is overwhelmingly human-driven: of the 14,960 conflicting internal merges, 96.1% are committed by a human and only 3.1% by the agent itself. Vendor practices vary nearly 60× in self-resolution rate, from Codex (0.5%) to Devin (29.4%); Claude Code is the only agent with a substantial agent-assisted share (24.1%), confirming an explicit pair-programming usage mode.

5.3 RQ2: How Do Agents Resolve Their Own Merges?

We now turn to the strategy distribution within the filtered set of 75,752 classified chunks (Table 1). Our analysis focuses on the 8,979 chunks resolved by agents withing agent-authored internal merges, while treating the 65,798 human-resolved chunks as a reference baseline; an additional 975 chunks were resolved through human-agent collaboration. We first present per-agent and pairwise contrasts (Section 5.3.1), then show that two of the five agents are each dominated by a single repository whose pattern shapes their entire profile (Section 5.3.2), and finally examine the asymmetry in unresolved-marker commits between agents and humans (Section 5.3.3).

Table 4: Resolution strategy distribution per resolver type. n is the number of resolved chunks. "Agents (autonomous)" aggregates the five per-agent rows; "Agents (assisted)" covers merges with a human committer plus an agent trailer (predominantly Claude Code's pair-programming usage; 743 of the 975 assisted chunks are on Claude Code PRs). The three groups together account for the 75,752 resolved chunks of Table 1. V is the bias-corrected Cramér's V effect size computed against the human reference row; higher V indicates greater divergence from the human strategy distribution. The 'NN' (None) strategy is omitted from this table as it accounts for a negligible fraction of the resolutions (e.g., 0.7% for humans).

Resolver	n	V_1	V_2	CC	CB	NC	V
OpenAI Codex	6,695	99.9%	0.1%	0.0%	0.0%	0.0%	0.32
Devin	1,235	19.8%	55.5%	5.2%	7.9%	11.5%	0.09
Copilot	776	15.9%	41.1%	5.5%	13.0%	23.5%	0.07
Cursor	165	21.8%	50.9%	2.4%	12.7%	11.5%	0.03
Claude Code	108	57.4%	36.1%	0.9%	1.9%	3.7%	0.02
Agents (autonomous)	8,979	79.7%	12.6%	1.2%	2.5%	3.9%	0.23
Agents (assisted)	975	36.5%	37.9%	6.4%	7.9%	11.4%	—
Humans	65,798	45.1%	26.0%	6.8%	9.9%	11.4%	—

5.3.1 Per-agent heterogeneity. We begin by comparing the three resolver groups at the aggregate level: autonomous agents (the union of all five), agent-assisted, and humans. Considered as a single block, autonomous agents (n = 8,979 chunks) differ from humans (n = 65,798) with Cramér's V=0.23, a moderate effect: agents lean far more heavily on V_1 (79.7% vs. 45.1% for humans) and produce noticeably fewer NC (3.9% vs. 11.4%) and CB (2.5% vs. 9.9%) resolutions. Agent-assisted resolutions (n = 975 chunks), by contrast, are almost indistinguishable from human behavior (V=0.032), with V_1 and V_2 nearly balanced at 36.5% and 37.8% respectively, and an NC share of 11.4% matching humans exactly. The contrast between autonomous agents and agent-assisted is itself substantial (V=0.30), revealing that the presence of human co-authorship and final commit decision dramatically reshapes the resolution strategy.

This aggregate reading, however, does not survive disaggregation. Decomposing the autonomous-agent block into its five constituents (top rows of Table 4) reveals that the agents do not share a common strategy. The V column reports the per-agent Cramér's V against the human reference, which ranges over more than an order of magnitude across the five agents. OpenAI Codex stands at one extreme with 99.9% V_1 on 6,695 chunks and a strong effect size (V=0.32), indicating its difference from the humans' distribution is large; this extreme uniformity is an outlier that Section 5.3.2 traces back to a single repository's bulk version-bump workflow. The other four agents differ substantially: Claude Code sits closest to humans on the V_1 axis (57.4% vs. 45.1%) with V=0.02, while Devin, Copilot, and Cursor invert the polarity and lean toward V_2 (V_2 between 41% and 56% vs. 45.1% for humans) with weak effect sizes between 0.03 and 0.09. The takeaway is that these four agents do not even share a common direction of deviation from humans; some over-pick V_1, others over-pick V_2, and all do so weakly, so they cannot be summarized by a single "agent profile." The moderate V=0.23 for the autonomous-agents block is therefore a property of corpus composition rather than a population-level claim: Codex's 6,695 chunks dominate the union by an order of magnitude over each of the other agents. Likewise, the agent-assisted row that appeared to "interpolate" between agents and humans is in fact dominated by Claude Code PRs (743 of 975 resolved chunks), consistent with Finding 1 that Claude Code is the only agent with a non-negligible explicit pair-programming usage.

Table 5: Pairwise strategy contrasts between agents. Effect size (V) measures the degree to which two agents' resolution strategies diverge from each other: Strong ($V \geq 0.30$) means the two agents resolve conflicts in substantially different ways; Moderate (0.10–0.30) indicates noticeable but not overwhelming divergence; Weak ($V < 0.10$) indicates near-identical strategy distributions. p_h is the Holm-corrected p-value (over 9 testable pairs). Gray text marks the only non-significant pair. † p-value not reported: chi-squared assumptions are violated (6 of 10 agent-strategy combinations have expected counts <5) due to the extreme sample-size imbalance (n=108 vs. n=6,695); V is still reported as a descriptive measure.

Agent pair	Effect size (V)	p_h
Copilot × OpenAI Codex	Strong (0.90)	$<10^{-15}$
Devin × OpenAI Codex	Strong (0.87)	$<10^{-15}$
Cursor × OpenAI Codex	Strong (0.85)	$<10^{-15}$
Claude Code × OpenAI Codex	Strong (0.60)	n.a.†
Claude Code × Cursor	Strong (0.37)	2.2×10^{-7}
Claude Code × Copilot	Strong (0.35)	1.1×10^{-21}
Claude Code × Devin	Moderate (0.24)	1.0×10^{-15}
Copilot × Devin	Moderate (0.20)	5.7×10^{-17}
Copilot × Cursor	Moderate (0.12)	0.006
Cursor × Devin	Weak (0.06)	0.074

Pairwise contrasts. Table 5 reports Cramér's V for the ten pairwise agent-against-agent strategy contrasts, with Holm-corrected p-values for the nine pairs where chi-squared assumptions hold. The Claude Code × Codex pair is excluded from significance testing: the extreme sample-size imbalance (n=108 vs. n=6,695) results in too few expected observations in 6 of the 10 possible agent-strategy combinations (expected count <5), violating chi-squared assumptions; so V=0.60 is reported as a descriptive measure only. Codex differs strongly from every other agent (V between 0.60 and 0.90), confirming its outlier status. Among the remaining four agents, Cursor and Devin are the only *statistically indistinguishable* pair (V=0.06, p_{HOLM}=0.074); the rest sit in the moderate range (V between 0.12 and 0.37) and remain significant after Holm correction.

File-category conditioning. Part of the per-agent heterogeneity is driven by file-type composition rather than resolution strategy alone. Table 6 reports the per-agent file-category breakdown of self-resolved chunks, which reveals distinct specializations: OpenAI Codex is 98.4% *config* (predominantly `package.json`); Copilot is 59.5% *lock* files (`pnpm-lock.yaml`, `package-lock.json`, `yarn.lock`); Devin is 61.1% *source* and 31.8% *lock*; Cursor is 78.8% *source*; Claude Code is 99.1% *source*. Within lock files, agents overwhelmingly select V_2 (Copilot 88%, Devin 84% of resolved lock-file chunks), consistent with the convention of accepting upstream-regenerated locks. Within source files the per-agent profiles largely

Table 6: File-category of self-resolved chunks per agent (%).

Agent	Source	Lock	Config	Generated	Doc	Migration
OpenAI Codex	0.1%	0.6%	98.4%	0.8%	0.0%	—
Copilot	32.2%	59.5%	5.3%	—	3.0%	—
Devin	61.1%	31.8%	4.7%	—	2.2%	0.2%
Cursor	78.8%	2.4%	4.8%	—	13.9%	—
Claude Code	99.1%	—	0.9%	—	—	—

converge. The V_2 bias reported in Table 4 for Copilot and Devin therefore reflects *file-type composition* in part, rather than differences in source-level merge behaviour.

Finding 2. The five agents do not share a common resolution approach. Per-agent contrasts against the human reference range from indistinguishable from humans (Claude Code, V=0.02) to strongly divergent (Codex, V=0.32); pairwise inter-agent contrasts span from indistinguishable (Cursor vs. Devin, V=0.06, the only pair to fail Holm-corrected significance) to nearly disjoint (Copilot vs. Codex, V=0.90). Part of this heterogeneity is driven by file-type composition: Copilot processes mostly lock files (59.5%) while Devin splits between source (61.1%) and lock files (31.8%), and lock files mechanically resolve V_2.

5.3.2 Single-repository concentration as a confound. The Codex outlier in Table 4 is not, in fact, a generic "Codex picks V_1" behaviour. Of the 61 Codex self-resolved merges, 55 (90.2%) come from a single repository, the `BlazingSu/nocobase` monorepo, and exhibit a near-identical profile: median 121 chunks per merge, 99.9% V_1, ∼92% one-line chunks. The dominant path within those merges is consistently `packages/core/app/package.json`: a single file edited in ∼ 121 regions, each of which is a one-line dependency-version bump (e.g., `"package-X": "^1.2.3"` on one side, a different version on the other), resolved by taking the agent's side (V_1) verbatim. These 55 merges have distinct SHAs that modify different version strings and so cannot be collapsed further by deduplication; they contribute 6,683 of Codex's 6,695 self-resolved chunks (99.8%), and roughly 74.4% of all agent-resolved chunks in the entire corpus.

The same pattern applies to Claude Code at smaller scale. Of the 12 Claude Code self-resolved merges, 10 (83.3%) come from a single repository, `itdojp/ITDO_ERP2`, and concentrate in three Python source files of an enterprise-resource-planning project (`test_users_extended.py`, `customer.py`, `budget.py`). Table 7 reports the top-1 and top-3 repository shares for each agent. Codex and Claude Code are the only two agents with top-1 share above 80%; the other three operate across 36–123 distinct repositories with no single repository contributing more than 10% of their merges. The interpretation is methodological: for agents with concentrated samples, the per-agent strategy distribution is a property of the dominant repository's workflow, not a population-level claim about the agent. The V=0.02 for Claude Code in Table 4 reflects 12 correlated merges, not 12 independent observations of agent behaviour, and we abstain from interpreting it as "Claude Code matches humans".

The corresponding fragility of any aggregate that involves the two concentrated agents is high. Removing only Codex's self-resolved chunks drops the aggregate agent-vs-human Cramér's V from 0.227 to 0.108 (weak); also removing bulk merges and lock / generated files brings it to 0.061. To rule out an "inherent property of agent-authored conflicts" explanation, we look at how *humans* resolve conflicts on Codex-authored PRs: the 48,617 resolved chunks committed by humans on Codex PRs follow the human reference almost exactly (V_1 44.0% vs 45.1%; V_2 25.7% vs 26.0%; CC 7.7% vs 6.8%; CB 10.1% vs 9.9%; NC 11.5% vs 11.4%). The Codex outlier is therefore a property of *Codex's resolution behaviour on the nocobase monorepo*, not of the conflicts those PRs produce.

Table 7: Repository concentration of self-resolved merges per agent. "Top-1 share" is the percentage of self-resolved merges that originate from the agent's most frequent repository.

Agent	Merges	Repos	Top-1	Top-3
OpenAI Codex	61	6	90.2%	95.1%
Claude Code	12	3	83.3%	100.0%
Cursor	43	36	9.3%	20.9%
Devin	224	123	5.4%	14.3%
Copilot	122	89	3.3%	9.8%

Finding 3. Two of the five agents are concentrated in a single repository: Codex (90.2% of self-resolved merges from `BlazingSu/nocobase`) and Claude Code (83.3% from `itdojp/ITDO_ERP2`). For these two agents, the per-agent distribution reflects the dominant repository's workflow rather than the agent itself. As a consequence, the per-agent conclusions about Claude Code and Codex in Finding 2, in particular, that Claude Code is indistinguishable from humans and that Codex is strongly divergent, are less generalizable than they may appear, since they derive predominantly from a single repository's patterns rather than from diverse agent behaviour.

5.3.3 Postponed: Agents Rarely Commit Unresolved Conflicts. Within the Imprecise bucket, a qualitatively distinct sub-category deserves separate attention: the 7,291 *Postponed* chunks (6.0% of all chunks; 15.9% of all Imprecise). For these, the localization algorithm did find a resolution region, but the committed file still contained conflict markers (<<<<<<<, =======, >>>>>>>), meaning the merge was committed without actually resolving the conflict, leaving syntactically broken code in the repository. The Postponed rate is strikingly asymmetric across resolver types: agent resolvers produced only 69 Postponed chunks out of 9,048 agent-resolved chunks (0.76%), compared to 7,190 out of 72,988 human-resolved chunks (9.85%), a roughly thirteen-fold difference. Among agents, the per-agent Postponed rate ranges from 0.58% (Devin) to 3.30% (Claude Code), all well below the human rate.

Finding 4. Agent resolvers commit *Postponed* chunks (files still containing conflict markers) at 0.76%, versus 9.85% for humans, a 13× difference. The 7,291 Postponed chunks (6.0% of all chunks) are syntactically broken code committed without resolving the conflict, most of them authored by humans; a CI-level grep would catch all of them.

6 Discussion

Our results carry implications for two audiences. For *researchers*, they argue against treating AI coding agents as a single class and expose how single-repository concentration can drive corpus-wide effect sizes in agent-attribution studies, with consequences for how empirical claims about AI behaviour should be reported. For *tool designers and pipeline operators*, they suggest where per-agent calibration matters more than global heuristics, where to focus resolver-assistance effort given the human-led nature of the task, and how the asymmetry between agents and humans on unresolved-marker commits opens cheap opportunities for CI-level safeguards. We discuss each audience in turn.

6.1 Implications for Researchers

Against treating AI as a single object. A growing literature reports performance numbers for "AI" on software-engineering tasks as if the underlying systems formed a single class [24, 27, 34, 39], possibly without distinguishing which agent (or which underlying model) produced each output. Our results push back against that framing in the specific case of merge-conflict resolution: even after fixing the task, the same human reference, and the same time period, the five agents we observed produce strategy distributions whose pairwise distance covers nearly the full possible range of Cramér's V (from indistinguishable to nearly disjoint). We acknowledge that the distinction between *agent* and *underlying language model* is itself a confound that we cannot fully control: the same agent may be backed by different model versions over time, and we have no per-merge visibility into which model handled each commit. But that confound *strengthens* rather than weakens the argument: if the observed heterogeneity is at least partly downstream of model choice, then aggregate "AI vs. humans" claims that paper over both the agent and the model layers will be even less stable than ours. For empirical AI-agent studies in software engineering, our recommendation is to report behaviour at least at the agent level, alongside the dominant repositories from which each agent's data is drawn (Section 5.3.2); we believe this is a precondition for any external-validity claim about "what AI does."

Methodological lesson on repository concentration. A secondary contribution of this paper is methodological. At first glance, the agent-resolved subset of 462 conflicting merges and 8,979 classifiable chunks looks adequate for a contingency analysis, but two of the five agents are dominated by a single repository: 90.2% of Codex's self-resolved conflicting merges come from `BlazingSu/nocobase` and 83.3% of Claude Code's from `itdojp/ITDO_ERP2` (Table 7). Without surfacing that concentration, the aggregate V=0.23 and the per-agent V=0.02 for Claude Code would both have read as population-level claims rather than as properties of single repositories. Two practices generalize beyond this study: (i) report the most active repository per analysis category when the study involves AI-agent attribution, since agentic workflows can repeat the same scripted pattern thousands of times within a single project, and (ii) stratify per-agent distributions by file category, since file-type composition (especially the share of lock files) drives a meaningful portion of the apparent strategy heterogeneity.

Asymmetric collaboration dynamics. Substantively, our data also indicates that human–AI collaboration on conflict resolution is *currently asymmetric*: agents author the branch, but humans resolve the conflicts inside it in 96.1% of cases, mirroring earlier observations about bot-authored PRs [4]. Devin (29.4% self-resolution on conflicting merges) and Claude Code (24.1% agent-assisted, plausibly a pair-programming mode) deviate sharply from this baseline; whether these collaboration patterns correlate with PR acceptance and post-merge defect rates is left to future work.

6.2 Implications for Tool Design

Per-agent calibration over global heuristics. Because the per-agent resolution profile varies by more than an order of magnitude in Cramér's V and is in part driven by what each agent worked on (Copilot processes mostly lock files and Devin a non-trivial share of them, where V_2 is the conventional resolution; Codex on its dominant repository collapsed to V_1 on `package.json` version bumps), merge-assistance tools whose evaluation benchmarks or default thresholds are derived from the union of all five agents will be miscalibrated for any individual one. Vendors and pipeline operators should track per-agent strategy distributions, condition on file category, and apply per-agent guardrails rather than treating the agent population as a single entity.

Targeted resolver augmentation. Because humans perform 96.1% of conflict resolutions and produce a richer strategy mix (11.4% NC, 9.9% CB), merge-assistance tools should be designed primarily for human reviewers; tools suggesting reference resolutions or flagging semantically suspicious verbatim picks add the most value. Conversely, the small agent-resolved share is highly predictable in some configurations (e.g., Codex on bulk monorepo updates), where a CI check that flags non-minimum-diff resolutions would catch silent regressions cheaply.

Conflict-complexity triage. The median conflicting merge contains only two chunks of 2–3 LOC on each side, well within the range that automated resolvers [10, 18, 19, 37] handle reliably; the heavy right tail justifies tiered escalation, with the extreme tail routed to human reviewers. The per-agent rates (Table 2) suggest tuning this triage per agent.

Agents rarely commit unresolved conflicts. Agents commit Postponed chunks in only 0.76% of resolutions, against 9.85% for humans, a 13× asymmetry. The likely explanation is that agents typically execute within scripted workflows that programmatically verify absence of markers before committing, whereas humans may commit under time pressure. Postponed commits are also the easiest failure mode to catch automatically (a grep for conflict markers suffices), yet the 6.56% human rate suggests this check is far from universal in CI pipelines for AI-authored PRs.

7 Threats to Validity

In this section, we discuss potential threats to the validity of our study and the steps taken to mitigate them. We follow the taxonomy by Wohlin et al. [41] to organize these threats into construct, internal, external, and conclusion validity.

Construct validity. Our pipeline replays three-way merges using `git mergefile` with the default `diff3` algorithm; projects with

custom merge drivers may therefore yield different conflict counts, a limitation shared with prior large-scale studies [7, 22, 32]. When a merge commit edits outside conflict boundaries, the LocalizeRes-Region anchor may fail and the chunk is reported as *Imprecise*; we report Imprecise rates explicitly. The classifier uses whitespace-normalized resolution text following Ghiotto et al. [22], a standard normalization in the literature [11, 21]. Our unit of analysis is the internal merge commit, so we do not observe resolutions performed via interactive rebase or branch abandonment; our results should thus be interpreted as a lower bound on observed in-PR conflict resolution activity. The path-category labels used in Section 5.3.2 (lock, generated, config, doc, source, migration) are basename- and prefix-based heuristics that do not capture all build artefacts in every ecosystem; consequently, file-type composition effects may be misestimated. The `packages/core/app/package.json` pattern that drives the Codex outlier illustrates that `package.json` can act as configuration in some monorepos and as a generated artefact in others.

Internal validity. Two attribution caveats apply to the per-agent partition of Table 4. First, we use the PR-authoring agent (from AIDev) as a proxy for the merge-committing agent: this is exact under the canonical workflow where each agent commits only inside its own bot-authored PRs, but inexact if a developer invokes a different agent inside an already-open PR. Second, the substring/suffix heuristic for the resolver (Section 4.5) introduces false positives (generic automation bots such as `kodiakhq[bot]` and `mergify[bot]`, and human logins containing matching substrings such as `devinkeehner`) and false negatives for agents committing under non-matching accounts. The dominant findings (Codex single-repo concentration, the order-of-magnitude per-agent V spread, the 13× Postponed asymmetry) are unlikely to be reversed under reasonable levels of such misclassification; however, smaller-n rows of Table 4 (Claude Code n=108, Cursor n=165, Copilot n=776) should be interpreted with this noise in mind.

External validity. Our scope is the public-GitHub portion of AIDev, so findings should not be extrapolated directly to private repositories. The agent distribution is heavily skewed toward Codex (approximately 76% of in-scope PRs); within the agent-resolved subset, the Codex slice is itself dominated by one repository (`BlazingSu/nocobase`), and we make no claim that other repositories using Codex would reproduce the same V_1 resolution collapse. The five agents are the most prominent as of mid-2025, and our baselines should be revisited as the ecosystem matures. Our pipeline reaches 57,582 of the approximately 116,000 AIDev repositories; the gap is primarily due to repositories or PR head references no longer accessible at fetch time (private, deleted, or with refs that GitHub no longer exposes). Within the reached repositories, 108 (0.2%) were further lost to cloning failures, and 236,449 merge SHAs across 88,339 PRs were unrecoverable for the same reason. We have no evidence of a systematic directional bias arising from these losses.

Conclusion validity. Conflict chunks exhibit intra-merge and intra-repository dependence, so p-values are anti-conservative (i.e., Type I error may be inflated). We therefore emphasize effect sizes (Cramér's V and Cliff's δ) and do not treat $p<0.05$ alone as evidence of practical importance. A more subtle threat is the differential Imprecise rate across resolver types: 16.86% for agent-resolved chunks versus 43.25% for human-resolved chunks, with two components: *localization failure* (16.1% vs. 33.4%) and *Postponed* cases (0.76% vs. 9.85%, Section 5.3.3). The higher localization success for agents is plausibly linked to their higher V_1 resolution rate, since V_1 resolutions exactly match one parent and are easier to anchor. This differential may inflate the agent's apparent V_1 dominance; however, the sensitivity analysis in Section 4.4, which reassigns Imprecise chunks under worst-case scenarios, bounds the agent-to-human V_1 ratio at 1.24×–2.45×, indicating that the observed per-agent contrasts are not artefacts of differential localization.

8 Conclusion

We presented the first large-scale empirical study of merge conflicts in AI-authored pull requests that attributes each resolution to a specific coding agent, recovering 50,700 internal merges (of which 14,960 conflicting) across 4,806 repositories from a 210,902-PR scope. Three findings carry the most operational weight. First, conflict resolution in agent-authored PRs is overwhelmingly human-led (96.1% of conflicting merges), and the per-agent self-resolution rate spans nearly 60×, justifying per-vendor rather than one-size-fits-all review and CI policies. Second, the five agents do not share a common resolution strategy: per-agent contrasts against the same human reference range over an order of magnitude in Cramér's V, and Cursor vs. Devin is the only indistinguishable pair among the ten. Third, two of the five agents are dominated by a single repository: 55 of the 61 Codex self-resolved merges are version-bump patterns from `BlazingSu/nocobase`, and 10 of the 12 Claude Code self-resolved merges come from `itdojp/ITDO_ERP2`. For these two agents, the per-agent strategy distribution reflects the dominant repository's workflow, not the agent itself, illustrating that single-repository concentration can drive corpus-wide effect sizes in AI-agent attribution studies.

As future work, a longitudinal analysis would establish whether the per-agent profiles and the single-repository concentration patterns documented here are stable as agent populations and underlying models evolve. A correctness assessment linking each resolution to downstream test outcomes would quantify the practical cost and implications of the per-agent strategy choices we report.

Artifact Availability

The replication package [13] and the collected dataset [12] are available: http://doi.org/10.5281/zenodo.22131603, MIT License and https://doi.org/10.6084/m9.figshare.32159415, CC BY 4.0 License.

Acknowledgements

This work was supported by the Brazilian National Council for Scientific and Technological Development (CNPq) under a Post-doctoral Fellowship (PDJ), Grant No. 155576/2025-9 and Productivity Grant No. 309300/2023-1; Fundação Carlos Chagas Filho de Amparo à Pesquisa do Estado do Rio de Janeiro (FAPERJ) for the grant E-26/204.145/2024; INES.IA (National Institute of Science and Technology for Software Engineering Based on and for Artificial Intelligence) (https://www.ines.org.br), CNPq grant 408817/2024-0. We acknowledge the use of Claude Code to speed up the implementation of scripts for data extraction and analysis. We carefully examined, tested, and often corrected all suggestions, whereby we take full responsibility for the form and content of the paper.

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] Alan Agresti. 2013. *Categorical Data Analysis* (3 ed.). Wiley, Hoboken, New Jersey.

[3] Waad Aldndni, Na Meng, and Francisco Servant. 2023. Automatic Prediction of Developers' Resolutions for Software Merge Conflicts. *Journal of Systems and Software* 206 (2023), 111836. doi:10.1016/j.jss.2023.111836

[4] Mahmoud Alfadel, Diego Elias Costa, Emad Shihab, and Mouafak Mkhallalati. 2021. On the Use of Dependabot Security Pull Requests. In *Proceedings of the 18th International Conference on Mining Software Repositories (MSR '21)*. IEEE, Madrid, Spain, 254–265. doi:10.1109/MSR52588.2021.00037

[5] Sven Apel, Jörg Liebig, Benjamin Brandl, Christian Lengauer, and Christian Kästner. 2011. Semistructured Merge: Rethinking Merge in Revision Control Systems. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering (ESEC/FSE '11)*. ACM, Szeged, Hungary, 190–200. doi:10.1145/2025113.2025141

[6] Wicher Bergsma. 2013. A bias-correction for Cramér's V and Tschuprow's T. *Journal of the Korean Statistical Society* 42, 3 (2013), 323–328. doi:10.1016/j.jkss.2012.10.002

[7] Alexander Boll, Yael van Dok, Manuel Ohrndorf, Alexander Schultheiß, and Timo Kehrer. 2024. Towards Semi-Automated Merge Conflict Resolution: Is It Easier Than We Expected?. In *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering (EASE '24)*. ACM, Salerno, Italy, 282–292. doi:10.1145/3661167.3661197

[8] Caius Brindescu, Iftekhar Ahmed, Rafael Leano, and Anita Sarma. 2020. Planning for Untangling: Predicting the Difficulty of Merge Conflicts. In *Proceedings of the 42nd IEEE/ACM International Conference on Software Engineering (ICSE '20)*. ACM, Seoul, South Korea, 801–811. doi:10.1145/3377811.3380308

[9] Yuriy Brun, Reid Holmes, Michael D. Ernst, and David Notkin. 2011. Proactive Detection of Collaboration Conflicts. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering (ESEC/FSE '11)*. ACM, Szeged, Hungary, 168–178. doi:10.1145/2025113.2025139

[10] Heleno de Souza Campos Junior, Gleiph Ghiotto L. de Menezes, Márcio de Oliveira Barros, André van der Hoek, and Leonardo Gresta Paulino Murta. 2026. Towards a Feasible Evaluation Function for Search-Based Merge Conflict Resolution. *ACM Transactions on Software Engineering and Methodology* 35, 5, Article 131 (April 2026), 35 pages. doi:10.1145/3748256

[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 36th Brazilian Symposium on Software Engineering (SBES '22)*. ACM, Maringá, Brazil, 425–434. doi:10.1145/3555228.3555229

[12] Heleno de Souza Campos Junior and Leonardo Gresta Paulino Murta. 2026. Agentic Conflicts: Dataset. https://doi.org/10.6084/m9.figshare.32159415. doi:10.6084/m9.figshare.32159415 Figshare.

[13] Heleno de Souza Campos Junior and Leonardo Gresta Paulino Murta. 2026. Agentic Conflicts: Replication Package (v1.0.0). https://doi.org/10.5281/zenodo.22131603. doi:10.5281/zenodo.22131603

[14] Guilherme Cavalcanti, Paulo Borba, Georg Seibt, and Sven Apel. 2019. The Impact of Structure on Software Merging: Semistructured versus Structured Merge. In *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering (ASE '19)*. IEEE, San Diego, CA, USA, 1002–1013. doi:10.1109/ASE.2019.00097

[15] Norman Cliff. 1993. Dominance statistics: Ordinal analyses to answer ordinal questions. *Psychological Bulletin* 114, 3 (1993), 494–509. doi:10.1037/0033-2909.114.3.494

[16] William G. Cochran. 1954. Some Methods for Strengthening the Common χ^2 Tests. *Biometrics* 10, 4 (1954), 417–451. doi:10.2307/3001616

[17] Jacob Cohen. 1988. *Statistical Power Analysis for the Behavioral Sciences* (2 ed.). Lawrence Erlbaum Associates, New Jersey.

[18] Elizabeth Dinella, Todd Mytkowicz, Alexey Svyatkovskiy, Christian Bird, Mayur Naik, and Shuvendu Lahiri. 2023. DeepMerge: Learning to Merge Programs. *IEEE Transactions on Software Engineering* 49, 4 (2023), 1599–1614. doi:10.1109/TSE.2022.3183955

[19] Jinhao Dong, Qihao Zhu, Zeyu Sun, Yiling Lou, and Dan Hao. 2023. Merge Conflict Resolution: Classification or Generation?. In *Proceedings of the 38th IEEE/ACM International Conference on Automated Software Engineering (ASE '23)*. IEEE, Luxembourg, Luxembourg, 1652–1664. doi:10.1109/ASE56229.2023.00155

[20] Olive Jean Dunn. 1964. Multiple Comparisons Using Rank Sums. *Technometrics* 6, 3 (1964), 241–252. doi:10.1080/00401706.1964.10490181

[21] Paulo Elias, Heleno de S. Campos Junior, Eduardo Ogasawara, and Leonardo Gresta Paulino Murta. 2023. Towards Accurate Recommendations of Merge Conflicts Resolution Strategies. *Information and Software Technology* 164 (2023), 107332. doi:10.1016/j.infsof.2023.107332

[22] 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

[23] Sture Holm. 1979. A Simple Sequentially Rejective Multiple Test Procedure. *Scandinavian Journal of Statistics* 6, 2 (1979), 65–70. http://www.jstor.org/stable/4615733

[24] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. 2024. SWE-bench: Can Language Models Resolve Real-World GitHub Issues?. In *Proceedings of the 12th International Conference on Learning Representations (ICLR 2024)*. https://openreview.net/forum?id=VTF8yNQM66

[25] Bakhtiar Khan Kasi and Anita Sarma. 2013. Cassandra: Proactive Conflict Minimization Through Optimized Task Scheduling. In *Proceedings of the 35th International Conference on Software Engineering (ICSE '13)*. IEEE, San Francisco, CA, USA, 732–741. doi:10.1109/ICSE.2013.6606619

[26] Olaf Leßenich, Janet Siegmund, Sven Apel, Christian Kästner, and Claus Hunsen. 2018. Indicators for Merge Conflicts in the Wild: Survey and Empirical Study. *Automated Software Engineering* 25, 2 (2018), 279–313. doi:10.1007/s10515-017-0227-0

[27] 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. arXiv:2507.15003 [cs.SE]

[28] Hao Li, Haoxiang Zhang, and Ahmed E. Hassan. 2026. AIDev: Studying AI Coding Agents on GitHub. arXiv:2602.09185. arXiv:2602.09185 [cs.SE]

[29] Tom Mens. 2002. A State-of-the-Art Survey on Software Merging. *IEEE Transactions on Software Engineering* 28, 5 (2002), 449–462. doi:10.1109/TSE.2002.1000449

[30] Nicholas Nelson, Caius Brindescu, Shane McKee, Anita Sarma, and Danny Dig. 2019. The Life-Cycle of Merge Conflicts: Processes, Barriers, and Strategies. *Empirical Software Engineering* 24, 5 (2019), 2863–2906. doi:10.1007/s10664-018-9674-x

[31] Daniel Ogenrwot and John Businge. 2026. AgenticFlict: A Large-Scale Dataset of Merge Conflicts in AI Coding Agent Pull Requests on GitHub. arXiv:2604.03551. arXiv:2604.03551 [cs.SE]

[32] Moein Owhadi-Kareshk, Sarah Nadi, and Julia Rubin. 2019. Predicting Merge Conflicts in Collaborative Software Development. In *Proceedings of the 13th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '19)*. IEEE, Copenhagen, Denmark, 292–301. doi:10.1109/ESEM.2019.8870173

[33] Rangeet Pan, Vu Le, Nachiappan Nagappan, Sumit Gulwani, Shuvendu Lahiri, and Mike Kaufman. 2021. Can Program Synthesis Be Used to Learn Merge Conflict Resolutions? An Empirical Analysis. In *Proceedings of the 43rd IEEE/ACM International Conference on Software Engineering (ICSE '21)*. IEEE, Madrid, Spain, 785–796. doi:10.1109/ICSE43902.2021.00077

[34] Sida Peng, Eirini Kalliamvakou, Peter Cihon, and Mert Demirer. 2023. The Impact of AI on Developer Productivity: Evidence from GitHub Copilot. arXiv:2302.06590. arXiv:2302.06590 [cs.SE]

[35] 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

[36] Chaochao Shen, Wenhua Yang, Minxue Pan, and Yu Zhou. 2023. Git Merge Conflict Resolution Leveraging Strategy Classification and LLM. In *Proceedings of the 23rd IEEE International Conference on Software Quality, Reliability, and Security (QRS '23)*. IEEE, Chiang Mai, Thailand, 228–237. doi:10.1109/QRS60937.2023.00031

[37] 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 (ESEC/FSE '22)*. ACM, Singapore, Singapore, 822–833. doi:10.1145/3540250.3549163

[38] 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

[39] Junjie Wang, Hao Huang, Yue Liu, Zixuan Xu, Xuan Hu, Lingming Yang, Zhenchang Xing, Xuming Yang, and Qing Wang. 2024. A Survey on Large Language Model-Based Agents for Software Engineering. arXiv:2409.02977. arXiv:2409.02977 [cs.SE]

[40] Edwin Bidwell Wilson. 1927. Probable Inference, the Law of Succession, and Statistical Inference. *J. Amer. Statist. Assoc.* 22, 158 (1927), 209–212. doi:10.1080/01621459.1927.10502953

[41] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, Anders Wesslén, et al. 2012. *Experimentation in Software Engineering*. Vol. 236. Springer, Berlin, Germany. doi:10.1007/978-3-642-29044-2