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

DANIEL OGENRWOT, University of Nevada Las Vegas, USA
JOHN BUSINGE, University of Nevada Las Vegas, USA

A Online Appendix: Dataset and Artifact Description

A.1 Overview

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, have received limited attention in this emerging 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 supports analysis at multiple levels of granularity:

- Pull request level (conflict occurrence and summary statistics)
- File level (conflicting files and conflict types)
- Region level (fine-grained conflict structure with line boundaries)
- Repository level (project metadata)

All entities are linked using a canonical identifier `pr_key` (derived from `repo_full_name` and `pr_number`). Each record also includes a `run_id` to ensure traceability across dataset extraction runs.

A.2 Artifact Availability

- **Dataset (Zenodo):** https://doi.org/10.5281/zenodo.19396916
- **Preprint (arXiv):** https://arxiv.org/abs/2604.03551
- **Code and Documentation:** https://github.com/unlv-evol/AgenticFlict
- **API Documentation:** https://agenticflict.readthedocs.io

A.3 RAW vs. CLEAN Dataset

The replication package provides two versions of the dataset:

RAW Dataset. The RAW dataset contains the full output of the extraction pipeline, including:

- Pipeline metadata (`run_id`, timestamps, versioning)
- Simulation parameters and intermediate states (`simulation_mode`, `simulation_anchor_time`)

Authors' Contact Information: 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.

- Debug and error information (`status_code`, `error_class`, `error_message_trunc`)
- Optional raw conflict content (configurable at extraction time)

This version supports reproducibility, debugging, and re-execution of the pipeline.

CLEAN Dataset. The CLEAN dataset is a processed, analysis-ready version derived from the RAW dataset. It:

- Removes pipeline-specific metadata and debug fields
- Excludes raw source code content to comply with GitHub Terms of Service
- Retains only analysis-relevant attributes and join keys
- Ensures consistency across tables via `pr_key`

A detailed field-level mapping is provided in `raw_to_clean_manifest.csv` (included in the Zenodo archive). Unless otherwise stated, all results reported in the paper are based on the CLEAN dataset.

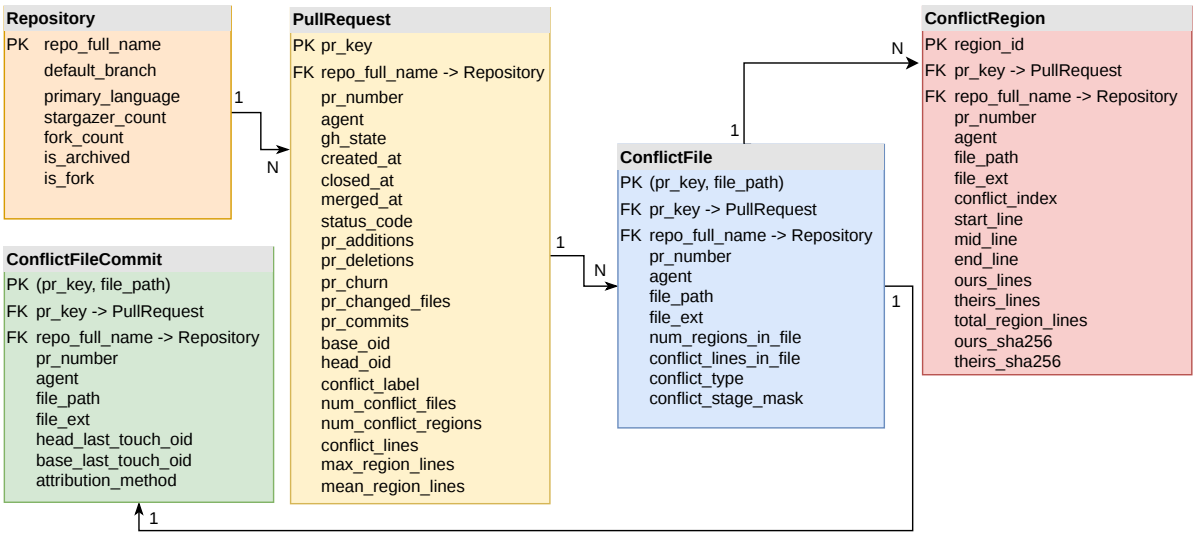

Fig. 1. Relational schema of the AgenticFlict CLEAN dataset. Tables are linked through the canonical `pr_key` identifier.

A.4 Dataset Files

The CLEAN dataset consists of the following tables:

- **agenticflict_pr_clean.csv** — Pull request-level metadata and conflict summary statistics. One row per simulated PR.
- **agenticflict_conflict_files_clean.csv** — File-level conflict information for each conflicting PR. One row per conflicting file.
- **agenticflict_regions_clean.csv** — Fine-grained conflict regions with line-level boundaries and cryptographic hashes. One row per conflict region.
- **agenticflict_conflict_file_commits_clean.csv** — Attribution of conflicting files to the most recent commits on each branch.
- **agenticflict_repo_clean.csv** — Repository-level metadata for all repositories in the dataset.

The RAW dataset contains the same tables with a `_raw` suffix, plus an additional table:

- **agenticflict_runlog_raw.csv** — Execution logs and failure tracking for every pipeline run.

A.5 Dataset Fields and Descriptions

Repository Table (`agenticflict_repo.csv`)

Field	Description
`repo_full_name`	Repository identifier (`owner/repo`)
`default_branch`	Default branch name
`primary_language`	Primary programming language
`stargazer_count`	Number of GitHub stars at extraction time
`fork_count`	Number of forks at extraction time
`is_archived`	Whether the repository is archived
`is_fork`	Whether the repository is itself a fork

Pull Request Table (`agenticflict_pr.csv`)

Field	Description
`pr_key`	Canonical identifier (`repo_full_name#pr_number`)
`repo_full_name`	Repository identifier (`owner/repo`)
`pr_number`	Pull request number within the repository
`agent`	AI coding agent that authored the PR
`gh_state`	GitHub PR state (`OPEN` or `CLOSED`)
`created_at`	PR creation timestamp (ISO 8601)
`closed_at`	PR closure timestamp (ISO 8601)
`merged_at`	PR merge timestamp; `NULL` for unmerged PRs
`pr_additions`	Number of lines added
`pr_deletions`	Number of lines deleted
`pr_changed_files`	Number of files modified
`pr_commits`	Number of commits in the PR
`pr_churn`	Total code churn (`additions + deletions`)
`base_oid`	Base branch commit SHA (from GitHub)
`head_oid`	Head branch commit SHA (from GitHub)
`simulation_mode`	Anchor strategy: `open_current` or `closed_base_at_close`
`conflict_label`	Boolean: `True` if merge simulation produced a textual conflict
`num_conflict_files`	Number of files containing conflict markers
`num_conflict_regions`	Total conflict regions across all files
`conflict_lines`	Total lines enclosed by conflict markers
`max_region_lines`	Lines in the largest single conflict region
`mean_region_lines`	Mean lines per conflict region

Conflict Region Table (`agenticflict_regions.csv`)

Field	Description
`region_id`	Unique identifier for the conflict region
`pr_key`	Foreign key to the PR table
`repo_full_name`	Repository identifier
`pr_number`	Pull request number
`agent`	AI coding agent
`file_path`	Relative path of the conflicting file
`file_ext`	File extension
`conflict_index`	0-based index of the region within the file
`start_line`	Line number of the `<<<<<` marker
`mid_line`	Line number of the `=======` separator
`end_line`	Line number of the `>>>>>` marker
`ours_lines`	Number of lines on the base (ours) side
`theirs_lines`	Number of lines on the incoming (theirs) side
`total_region_lines`	Total lines in the conflict region
`ours_sha256`	SHA-256 hash of the `ours` conflict block
`theirs_sha256`	SHA-256 hash of the `theirs` conflict block

Conflict File Table (`agenticflict_conflict_files.csv`)

Field	Description
`pr_key`	Foreign key to the PR table
`repo_full_name`	Repository identifier
`pr_number`	Pull request number
`agent`	AI coding agent
`file_path`	Relative path of the conflicting file
`file_ext`	File extension
`num_regions_in_file`	Number of conflict regions in this file
`conflict_lines_in_file`	Total lines enclosed by conflict markers in this file
`conflict_type`	Git 3-way merge classification (e.g., `both_modified`, `both_added`)
`conflict_stage_mask`	Encoded unmerged index stage information

Conflict File Commit Table (`agenticflict_conflict_file_commits.csv`)

Field	Description
`pr_key`	Foreign key to the PR table
`repo_full_name`	Repository identifier
`pr_number`	Pull request number
`agent`	AI coding agent
`file_path`	Relative path of the conflicting file
`file_ext`	File extension
`head_last_touch_oid`	SHA of the last commit to touch this file on the head branch
`base_last_touch_oid`	SHA of the last commit to touch this file on the base branch
`attribution_method`	Method used for attribution (`git_log_last_touch`)

B Extraction Pipeline and Reproducibility

B.1 Pipeline Overview

The extraction pipeline processes each Agentic PR through seven sequential steps:

(1) **Source loading.** Agentic PRs are loaded from the `hao-li/AIDev` HuggingFace dataset, which catalogues AI agent-authored pull requests.
(2) **Metadata retrieval.** Repository and PR metadata are fetched via the GitHub GraphQL API, including commit OIDs, PR state, and repository attributes.
(3) **Filtering.** Merged PRs are excluded from the dataset; only `OPEN` and `CLOSED`-unmerged PRs are retained.
(4) **Repository caching.** Repositories are cloned locally using blobless shallow clones (`-filter=blob:none`) and cached on disk.
(5) **Simulation anchor selection.** For `OPEN` PRs, the current `base_oid` and `head_oid` are used. For `CLOSED`-unmerged PRs, the base branch state at `closed_at` is resolved via `git rev-list`.
(6) **Merge simulation.** A local `git merge -no-commit -no-ff` is executed using the anchored commit OIDs.
(7) **Conflict extraction.** Conflict markers (`«««««/=======/»»»»»`) are parsed to extract region-level metrics and SHA-256 hashes.

B.2 Reproducing a Single Label

Each PR row stores `base_oid` and `head_oid`. Any conflict label can be independently verified from public GitHub history:

```
# Clone the repository
git clone https://github.com/{owner}/{repo} /tmp/verify-repo
cd /tmp/verify-repo

# Fetch the required commits and run the merge simulation
git fetch origin {head_oid}
git checkout {base_oid}
git merge --no-commit --no-ff {head_oid}

# Non-zero exit code and conflict markers confirm a conflicting PR
```

Listing 1. Verifying a single conflict label

C Loading and Using the Dataset

C.1 Requirements

- Python 3.9 or higher
- `pandas` (`pip install pandas`)

C.2 Loading the CLEAN Dataset

```python
from pathlib import Path
import pandas as pd

DATA_DIR = Path("data/clean")

pr      = pd.read_csv(DATA_DIR / "agenticflict_pr_clean.csv")
regions = pd.read_csv(DATA_DIR / "agenticflict_regions_clean.csv")
cfiles  = pd.read_csv(DATA_DIR / "agenticflict_conflict_files_clean.csv")
commits = pd.read_csv(DATA_DIR / "agenticflict_conflict_file_commits_clean.csv"
    )
repo    = pd.read_csv(DATA_DIR / "agenticflict_repo_clean.csv")

print(f"PRs: {len(pr):,}  |  Conflicting: {pr['conflict_label'].sum():,}")
```

Listing 2. Loading the CLEAN dataset

C.3 Loading the RAW Dataset

```python
from pathlib import Path
import pandas as pd

DATA_DIR = Path("data/raw")

pr      = pd.read_csv(DATA_DIR / "agenticflict_pr_raw.csv")
regions = pd.read_csv(DATA_DIR / "agenticflict_regions_raw.csv")
cfiles  = pd.read_csv(DATA_DIR / "agenticflict_conflict_files_raw.csv")
commits = pd.read_csv(DATA_DIR / "agenticflict_conflict_file_commits_raw.csv")
repo    = pd.read_csv(DATA_DIR / "agenticflict_repo_raw.csv")
runlog  = pd.read_csv(DATA_DIR / "agenticflict_runlog_raw.csv")

print(f"PRs: {len(pr):,}  |  Run log entries: {len(runlog):,}")
```

Listing 3. Loading the RAW dataset

C.4 Example Analyses

```python
agent_stats = (
    pr.groupby("agent")["conflict_label"]
```

```
    .agg(total="count", conflicts="sum")
    .assign(conflict_rate=lambda d: d["conflicts"] / d["total"])
    .sort_values("conflict_rate", ascending=False)
)
print(agent_stats)
```

Listing 4. Conflict rate by AI agent

```
merged = regions.merge(
    pr[["pr_key", "agent", "repo_full_name"]], on="pr_key"
)
print(merged.groupby("agent")["total_region_lines"].describe())
```

Listing 5. Joining PR metadata with conflict regions

C.5 Notes

- The CLEAN dataset is recommended for analysis and modeling.
- The RAW dataset includes additional pipeline metadata and execution logs useful for debugging and reproducibility audits.
- All tables can be joined using `pr_key` as the primary identifier.
- Repository metadata reflects values at extraction time and may differ from current repository state.

D Dataset Usage

AgenticFlict is designed to support, but is not limited to, the following research directions:

- **Conflict prediction.** Train classifiers that predict whether an incoming agent PR will conflict, using PR size, repository activity, file overlap, and agent identity as features.
- **Agent behavior comparison.** Compare conflict profiles (rate, severity, type) across the five included AI agents to identify behavioral patterns that elevate integration risk.
- **Automated conflict resolution.** Use paired conflict-region data (including SHA-256 hashes of both sides of each conflict) as inputs to resolution models.
- **Software repository mining.** Study how repository characteristics (language, stars, fork count) moderate conflict likelihood in AI-authored contributions.
- **Longitudinal analysis.** Use `created_at` and `closed_at` timestamps to analyze how agent deployment patterns and conflict rates have changed over time.

E Limitations

- **Textual conflicts only.** The dataset captures syntactic merge conflicts as detected by Git's 3-way merge. Semantic conflicts (syntactically correct but behaviorally incompatible changes) are not detected.
- **Excluded PRs.** PRs are excluded when repositories are deleted, made private, or inaccessible at extraction time, introducing potential survivorship bias toward more active repositories.
- **No human-authored baseline.** The current version focuses exclusively on AI agent PRs. A matched human-authored baseline will be added in a future release to enable direct comparison.
- **Snapshot in time.** Conflict labels for `OPEN` PRs reflect repository state at extraction time. Re-running the pipeline at a later date may yield different results as base branches advance.

- **Agent coverage.** Coverage is limited to agents represented in the `hao-li/AIDev` source dataset. Agents with few PRs may exhibit higher variance in per-agent metrics.