A Dataset of Agentic AI Coding Tool Configurations

Matthias Galster
University of Bamberg
Germany
mgalster@ieee.org

Seyedmoein Mohsenimofidi
Heidelberg University
Germany
s.mohsenimofidi@uni-heidelberg.de

Levi Böhme
University of Bayreuth
Germany
levi.boehme@uni-bayreuth.de

Jai Lal Lulla
Singapore Management University
Singapore
jailal.l.2025@phdcs.smu.edu.sg

Muhammad Auwal Abubakar
University of Bamberg
Germany
muhammad.abubakar@uni-bamberg.de

Christoph Treude
Singapore Management University
Singapore
ctreude@smu.edu.sg

Sebastian Baltes
Heidelberg University
Germany
sebastian.baltes@uni-heidelberg.de

Abstract

Developers increasingly rely on agentic AI coding tools such as Claude Code and OpenAI Codex, which autonomously plan, execute, and iterate on coding tasks. To steer these tools, developers create repository-level configuration artifacts (e.g., Markdown files) for configuration mechanisms such as Context files, Skills, Rules, and Hooks. There is no curated dataset that captures these configurations at scale. We address this gap by presenting a dataset of agentic AI coding tool configurations collected from open-source GitHub repositories. We selected 40,585 actively maintained repositories through metadata filtering, classified them using GPT-5.2 to identify 36,710 engineered software projects, and systematically detected configuration artifacts in these repositories. The dataset covers 4,741 repositories across five tools (Claude Code, GitHub Copilot, OpenAI Codex, Cursor, Gemini) and eight configuration mechanisms. We collected 15,612 configuration artifacts, the full content of 45,126 configuration files associated with configuration artifacts, and 148,551 AI-co-authored commits. The dataset and the complete construction pipeline are publicly available on Zenodo under CC BY 4.0. An interactive website allows researchers to browse and explore the data. This data supports research on context engineering, tool adoption patterns, and human-AI collaboration.

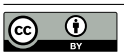
CCS Concepts

• **Software and its engineering**;

Keywords

Software Engineering, Generative AI, AI Agents, Configuration

ACM Reference Format:
Matthias Galster, Seyedmoein Mohsenimofidi, Levi Böhme, Jai Lal Lulla, Muhammad Auwal Abubakar, Christoph Treude, and Sebastian Baltes. 2026. A Dataset of Agentic AI Coding Tool Configurations. In *3rd ACM International Conference on AI-powered Software (AIware 2026), July 6–7, 2026, Montreal, Canada.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/XXXXXXX.XXXXXXX



AIware 2026, Montreal, Canada

ACM ISBN 979-8-4007-XXXX-X/2026/04
https://doi.org/10.1145/XXXXXXX.XXXXXXX

1 Introduction

Agentic AI coding tools have changed how developers write software [7]. Tools such as OpenAI Codex, Claude Code, and Cursor operate with greater autonomy than traditional AI coding assistants: they plan multi-step tasks, execute code, run tests, and iterate on results with minimal human intervention [5, 12]. These tools break down high-level goals into subtasks and adjust their approach based on real-time feedback [17]. For instance, when asked to implement a new feature, such a tool may identify the relevant parts of the codebase, apply the necessary modifications, run tests, and refine its output based on the results.

As agentic tools see growing adoption [16], developers have started to configure their behavior through *configuration mechanisms*, i.e., means by which developers tailor tool and agent behavior to a project or workflow [6]. Prior work [6] identified and defined eight configuration mechanisms as follows: Context files are Markdown files providing persistent context loaded every session; Skills bundle reusable knowledge and invocable workflows; Subagents are specialized agents running in an isolated context; Commands are user-triggered shortcuts for predefined prompts; Rules are system-level instructions to control agent behavior; Settings are JSON/TOML configurations for project-level tool behavior; Hooks are scripts executed at specific agent lifecycle points; and MCP configurations register external tool or data connections via the Model Context Protocol.

A *configuration artifact* is a tangible instance of a mechanism [6]: a single file (e.g., `CLAUDE.md`) or a directory containing multiple *configuration files* (e.g., `.claude/agents/` with one Markdown file per subagent). Through configuration artifacts, developers communicate project constraints, coding conventions, and architectural decisions to AI agents [3, 13]. These artifacts are version-controlled and collaboratively maintained. They represent a new category of

software engineering documentation written for AI agents rather than human teammates, functioning as specifications that encode constraints and expected behaviors to guide code generation and modification.

Despite this rapid adoption, no curated dataset systematically captures how developers configure agentic AI coding tools across open-source repositories. Without such data, researchers cannot study questions such as which configuration mechanisms developers actually use, how configurations differ across tools and programming languages, or how AI-assisted contributions relate to project characteristics or impact project quality.

We address this gap with a dataset of agentic AI coding tool configurations collected from GitHub. This work builds on a previous study [6], which identified a taxonomy of eight configuration mechanisms and characterized their adoption across 2,923 repositories. Here, we contribute the underlying data as a reusable research artifact with expanded scope and new data types: (1) a curated sampling frame of 40,585 actively maintained GitHub repositories with rich metadata; (2) LLM-based classification of 39,426 repositories as *engineered* software projects, including justifications for the classification; (3) 15,612 configuration artifacts from 4,741 repositories across five tools and eight configuration mechanisms, together with the full content of 45,126 configuration files; (4) 148,551 AI-co-authored commits from repositories that use agentic AI coding tools; and (5) a fully reproducible construction pipeline and an interactive exploration website.

2 Related Work

Work related to this dataset can be categorized into studies of AI-assisted development behavior, analyses of configuration artifacts, and existing datasets on AI-generated code.

Robbes et al. [16] measured the adoption of coding agents on GitHub through pull request analysis, focusing on code contributions rather than configuration practices. Watanabe et al. [19] and Horikawa et al. [9] studied AI-authored pull requests and agentic refactoring, respectively. Both examined what AI tools produce, not how developers configure them. He et al. [8] analyzed Cursor's effect on development velocity and code complexity for a single tool without examining configuration artifacts. Jiang and Nam [10] studied Cursor rule files, covering one configuration mechanism for one tool. These studies produced empirical findings but not reusable, multi-tool datasets.

Two studies target configuration artifacts directly. Chatlatanagulchai et al. [3] conducted an empirical study of `AGENTS.md` context files, a single artifact type. Mohsenimofidi et al. [13] studied context engineering practices across open-source repositories, analyzing the textual content of context files. Our companion study [6] defined a taxonomy of eight configuration mechanisms across five tools and analyzed their adoption in 2,923 repositories. That work produced findings and published supplementary scripts and CSV summaries for 2,923 repositories. This paper contributes an expanded, standalone dataset with 4,741 repositories, new data types not present in the supplementary material (148,551 AI-co-authored commits, 45,126 raw configuration files with full text content, per-artifact git metadata), and an improved pipeline to classify *engineered* repositories that reduced unsure cases (i.e., repositories that could not confidently be classified as engineered) by 93%.

Several related datasets capture other aspects of AI-assisted development. The MSR 2026 Mining Challenge dataset (AIDev) [12] provides 932,791 agent-authored pull requests across five AI tools, focusing on code contributions rather than configurations. AgentPack [21] collects 1.8M code edits co-authored by AI agents and humans, targeting model training rather than repository-level adoption analysis. Xiao et al. [20] mine explicit self-admissions of GenAI usage from commits and documentation, providing high-precision but inherently incomplete signals since many projects do not disclose AI tool usage. Repository sampling approaches such as the SEART GitHub Search (GHS) tool [4] and the criteria of Munaiah et al. [14] for identifying engineered projects provide foundations for data collection, but no existing dataset applies them to AI coding tool configurations specifically. To our knowledge, no dataset provides multi-tool AI coding tool configurations with associated metadata, temporal information, and raw artifact contents.

3 Dataset Overview

The dataset is organized around three nested levels of repositories. At the broadest level, a *sampling frame* of 40,585 GitHub repositories provides metadata for the general population of actively maintained open-source projects. We were able to classify a subset of 39,426 repositories along our definition of engineered software projects. At the narrowest level, our dataset contains 4,741 repositories with detected AI coding tool configurations, corresponding metadata, raw configuration file contents, and AI-co-authored commit histories. Table 1 lists all data files with row counts. We describe each level below.

Sampling frame. All 40,585 repositories include GitHub metadata: primary programming language, star and fork counts, commit totals, contributor counts, license, creation date, and repository topics. The ten programming languages in the sampling frame are C, C#, C++, Go, Java, JavaScript, PHP, Python, Rust, and TypeScript. Among the 4,741 repositories with detected configurations, TypeScript dominates (24.5%), followed by Python (18.7%), Go (13.8%), Java (8.0%), and C# (7.5%). This baseline allows researchers to compare tool-adopting repositories against the broader population and to control for language-specific effects.

Classification. Of the 40,585 repositories, 1,159 were excluded before classification because they had become unavailable (15), lacked a README file (74), or had a non-English README (1,070). The remaining 39,426 repositories carry classification labels produced by GPT-5.2, indicating whether each repository is an engineered software project. Each label includes a justification text. The classification identified 36,710 engineered projects, 2,564 non-engineered repositories, and 152 unsure cases. We discuss this process in more detail in Section 4.2.

Configuration artifacts. Among the engineered projects, 4,741 contain at least one agentic AI coding tool configuration artifact. The dataset covers five tools (Claude Code, GitHub Copilot, OpenAI Codex, Cursor, and Gemini) and eight configuration mechanisms that we introduced in Section 1: Context files, Skills, Subagents,

Table 1: Dataset files and row counts. Large files are distributed as 7z archives.

File	Rows	Description
`repos.csv`	40,585	Sampling frame with metadata
`commits.csv`	148,551	AI-co-authored commits
`context_files.csv`	9,491	Context files
`skills.csv`	2,430	Skills
`commands.csv`	1,098	Commands
`rules.csv`	997	Rules
`subagents.csv`	884	Subagents
`settings.csv`	472	Settings
`mcp.csv`	138	MCP server configurations
`hooks.csv`	102	Hooks
`repos_data.7z`	–	45,126 raw configuration files

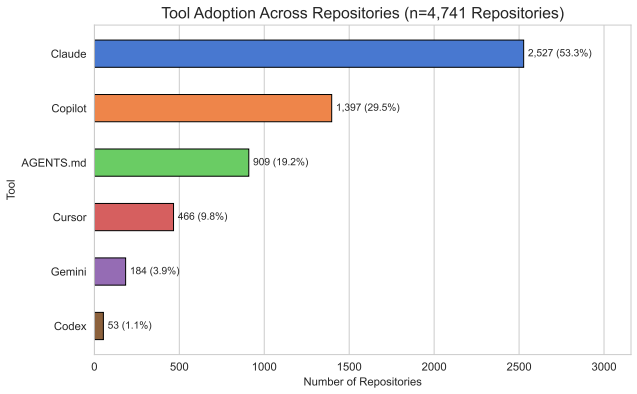


Figure 1: Number of repositories with detected configuration artifacts per agentic AI coding tool.

Commands, Rules, Settings, Hooks, and MCP configurations. Figure 1 shows the number of repositories per tool: Claude Code leads with 2,527 repositories, followed by Copilot (1,397) and Cursor (466). An additional 909 repositories contain only an `AGENTS.md` file without any tool-specific configuration, suggesting use of a vendor-neutral convention. Figure 2 shows the distribution of artifacts across types. Context files are the most common (found in 4,466 repositories), while Hooks (101) and MCP configurations (124) remain rare.

Figure 3 breaks this down further by showing which configuration mechanisms are used with which tools. Context files are near-universal across all tools, but advanced mechanisms concentrate in specific tools: Skills, Subagents, and Commands appear almost exclusively in Claude Code repositories, while Rules are specific to Cursor. This cross-tabulation, derived directly from the dataset, illustrates the tool-specific configuration cultures forming in practice. Figure 4 shows which combinations of configuration mechanisms co-occur: the majority of repositories use only Context files, while smaller groups combine Context files with Settings, Skills, or multiple advanced mechanisms.

Per-artifact CSV files record the file path, git creation date, commit count, and first and last commit SHAs for each artifact. Context

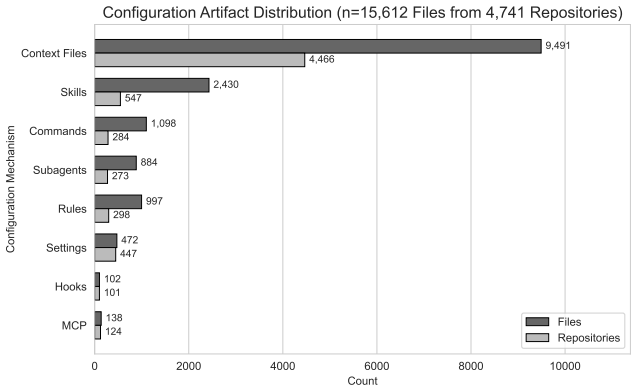


Figure 2: Number of configuration artifacts by type. For each mechanism, the chart shows the total file count and the number of repositories containing at least one artifact.

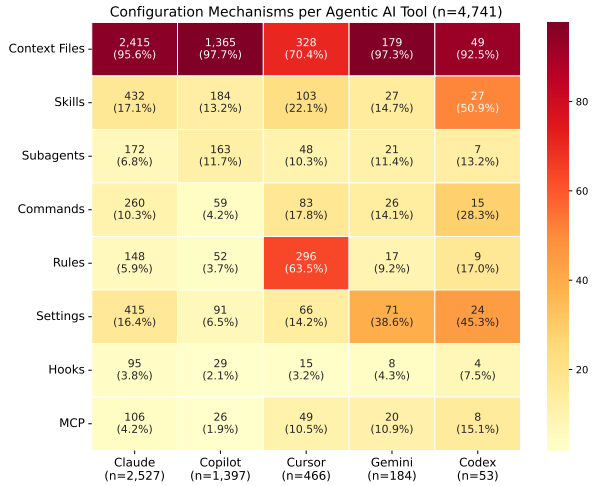


Figure 3: Configuration mechanisms per tool. Each cell shows the count and percentage of repositories using a given tool that also adopt the corresponding mechanism.

files include additional metadata: detected natural language, line count, whether the file references another file, and AI authorship information for the file's commits. The archive `repos_data.7z` contains the full text of all 45,126 collected configuration files, enabling qualitative content analysis and model training. All 15,612 artifacts across all eight mechanism types were validated to be non-empty.

AI-co-authored commits. For every repository with at least one configuration artifact, we collected all AI-co-authored commits (see Section 4.3 for the detection method). The resulting `commits.csv` contains 148,551 commits from 3,394 repositories, each with a timestamp, full commit message, and the detected AI tool. For Context files, the per-artifact CSV additionally records whether the file's creation commit was AI-co-authored and the total number of AI-co-authored commits that modified the file. This

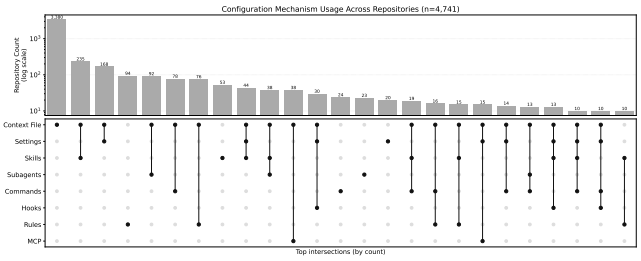


Figure 4: Co-occurrence of configuration mechanisms across repositories. The bar chart shows how many repositories use each combination; the matrix below indicates which mechanisms are included.

allows researchers to study how AI tools participate in creating and maintaining their own configuration artifacts.

4 Dataset Construction

The dataset was constructed through a three-stage pipeline: repository sampling, LLM-based classification, and configuration artifact extraction. Figure 5 illustrates the pipeline with counts at each stage.

4.1 Repository Sampling

We used the GHS dataset [4][1] as a starting point. GHS provides a regularly updated snapshot of GitHub repository metadata collected via the GitHub API. At the time of our query, it contained over one million repositories with metadata covering repository information, activity metrics, and social engagement indicators.

Our baseline query was designed to avoid common pitfalls in GitHub mining [11]. A repository in our sample had to (1) have a license, (2) not be a fork, (3) have at least two contributors, (4) have at least one pull request, (5) be at least 18 months old, and (6) have had a commit within the past six months. GHS additionally includes only repositories with at least ten stars. Requiring a license enables downstream open-source filtering. Excluding forks avoids duplicated codebases. The contributor and pull request thresholds exclude toy and solo projects. The age and recency requirements filter both immature and stale repositories.

The query, executed on 2026-03-16, returned 187,304 repositories. We then applied metadata filtering stages (Table 2): (1) removing archived, disabled, or locked repositories; (2) retaining only OSI-approved software licenses per the SPDX License List; (3) keeping only the ten most common programming languages; (4) excluding educational and resource-aggregation repositories identified by name and topic patterns; (5) applying median-based lower thresholds on commits (≥352) and watchers (≥6) [13]; (6) and deduplicating redirected repositories. After filtering, 40,585 repositories remained.

4.2 Repository Classification

We classified the sampled repositories to distinguish engineered software projects from non-software or toy repositories. After cloning each repository, we excluded 15 that were unavailable, 74 with missing or empty README files, and 1,070 with non-English READMEs (detected using `lingua-language-detector`), leaving 39,426 repositories.

[1] https://seart-ghs.si.usi.ch/

Table 2: Metadata filtering stages and their impact on dataset size, applied to the baseline GHS query result of 187,304 repositories.

Filtering Step	Δ Removed	Remaining
Initial Dataset	–	187,304
(1) Archived / Disabled / Locked	2,241	185,063
(2) Non-OSI / Excluded Licenses	24,646	160,417
(3) Non-Top-10 Languages	31,252	129,165
(4) Educational / Resource Repos	1,994	127,171
(5) Commits < 352 ∨ Watchers < 6	86,331	40,840
(6) Duplicate Repos	255	40,585
Metadata-Filtered Dataset	–	**40,585**

For each repository, we assembled three types of inputs for the classification: (1) the README content, (2) a summary of the file structure produced by GitHub Linguist, and (3) summaries of external links referenced in the README.

For the file structure, we ran GitHub Linguist on each repository and summarized the output by retaining the top seven languages per repository (ranked by byte count; covering the 75th percentile of the language-per-repository distribution) and up to three representative file paths per directory level, capped at depth eight (where 75% of all files across our sample reside). This keeps the representation informative without exceeding the context window of the model used for classification (see below).

For external links, we extracted all URLs from the README files (883,997 links across 39,426 repositories). We retained only those returning HTTP 200 with `text/html` content type (565,255 links) and summarized the first 24 per repository (the 75th percentile of links per repository) using GPT-5-nano. Prana et al. [15] found that links in README files cluster in the "What" and "How" sections that typically appear near the top of the file, so earlier links tend to be more informative about the repository's purpose.

We used GPT-5.2 via the OpenAI Batch API to classify each repository based on these three inputs. The classification prompt was based on Munaiah et al.'s definition of engineered software projects [14], i.e., projects that show clear evidence of software engineering practices such as stated purpose, installation instructions, documentation, testing, or CI/CD. Two authors spot-checked the assigned labels on a random sample. We piloted both GPT-5.2 and GPT-5-nano; GPT-5.2 showed better adherence to the classification criteria. The exact classification prompt is included in the pipeline archive [2].

The classification labeled 36,710 repositories as engineered, 2,564 as non-engineered, and 152 as unsure. Compared to our companion study [6], which classified repositories based on README content alone, this improved pipeline reduced unsure cases from 2,204 to 152. The reduction is attributable to the additional context from file

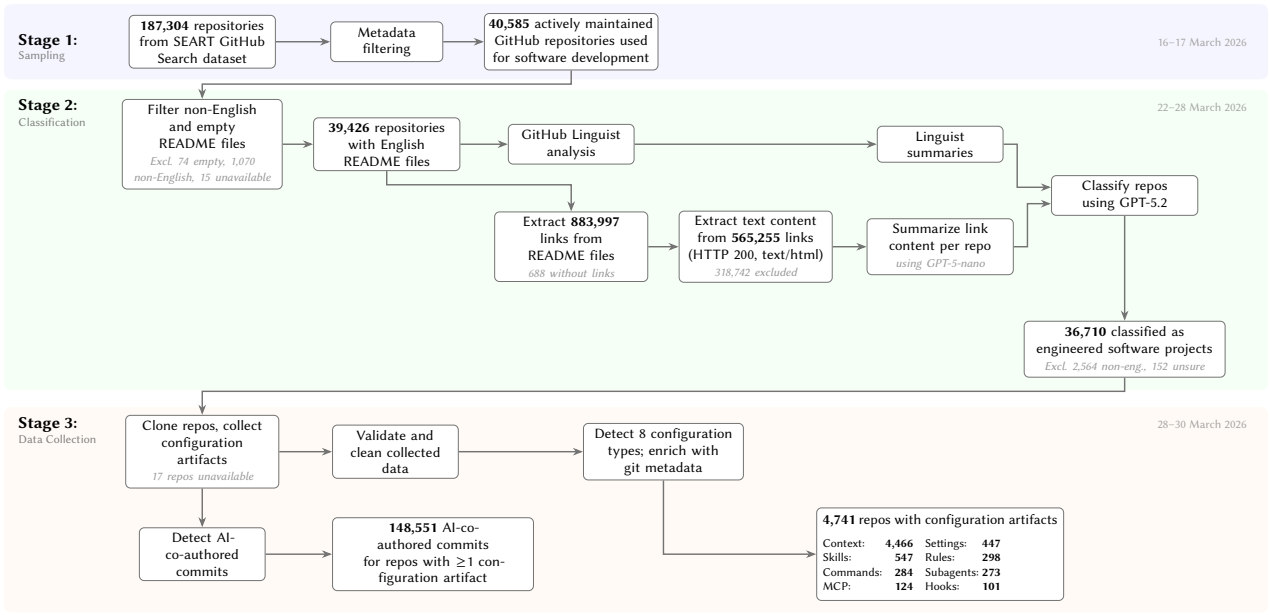


Figure 5: Three-stage data collection pipeline: repository sampling from the SEART GitHub Search dataset, LLM-based classification of engineered software projects, and extraction of AI coding tool configuration artifacts and AI-co-authored commits.

structure summaries and external link summaries, which helped resolve borderline cases.

4.3 Configuration Artifact Extraction

We selected five agentic AI coding tools based on the 2025 Stack Overflow Developer Survey [18]: Claude Code, GitHub Copilot, OpenAI Codex, Cursor, and Gemini. The survey asked developers which AI agent tools they use; we selected the four most popular tools and substituted Codex for ChatGPT, as Codex is the agentic coding tool by the same vendor. We included Cursor given its prominence in recent software engineering research [8, 10].

For each tool, we documented the repository-level files and directories that indicate its presence and developed detection heuristics based on this documentation [6]. We detect the eight configuration mechanisms introduced in Section 1: Context files (e.g., `CLAUDE.md`, `AGENTS.md`, `.cursorrules`), Skills, Subagents, Commands, Rules, Settings, Hooks, and MCP configurations. For GitHub Copilot, Cursor, and Gemini, the detected files apply to both their conversational and agentic interfaces.

We cloned the 36,710 engineered repositories and applied these heuristics to extract configuration artifacts. After validation and cleaning (removing invalid file paths, recalculating detection flags, and excluding repositories where no valid artifacts remained), 4,741 repositories contained at least one artifact. We enriched each artifact with git metadata: creation date, total commit count, and first and last commit hashes. For Context files, we additionally detected the natural language, identified files that reference other files, and attributed AI authorship at the commit level.

To assess the extent of active AI tool use in these repositories, we scanned each repository's full commit history for AI-co-authored commits. Detection uses regex-based heuristics across three scopes, applied to every commit reachable from any branch. First, *author identity* patterns match tool-specific bot accounts in the author and committer name and email fields (e.g., `copilot-swe-agent[bot]`, `claude[bot]`). Second, *git trailer* patterns match structured attribution lines such as `Co-authored-by:` and `Generated-by:`, extracted by walking the commit body in reverse to isolate the trailer block from the message body. Third, *message body* patterns match attribution phrases and tool-specific branch prefixes in merge commits. The patterns cover five tools (Claude, Copilot, Cursor, Codex, and Gemini) and are documented with unit tests in the pipeline archive [2]. This produced 148,551 AI-co-authored commits across 3,394 of the 4,741 configured repositories. This confirms that the detected configuration artifacts correspond to active tool use rather than idle presence. The commit data also reveals how AI tools participate in creating and maintaining their own configuration artifacts: for example, researchers can trace whether a `CLAUDE.md` file was initially authored by a human or by an AI tool, and how the file evolved over time through human and AI contributions.

5 Use Cases

The dataset supports a range of empirical research questions. We describe concrete directions and illustrate each with the specific data fields that enable it.

Tool adoption and evolution. Each configuration artifact in the dataset carries a `created_at` timestamp and commit count, and each AI-co-authored commit has a timestamp. Researchers can use these fields to study when projects adopted specific tools, in what order they added configuration mechanisms, and whether tool adoption correlates with changes in commit activity. Figure 6 shows the cumulative number of repositories adopting each artifact type,

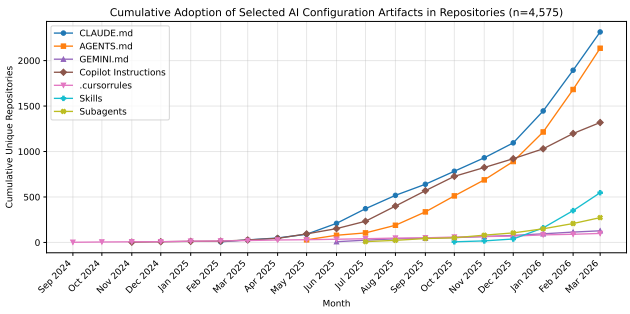


Figure 6: Cumulative adoption of selected AI configuration artifacts over time. Each curve shows the cumulative number of repositories in which the artifact type was first observed.

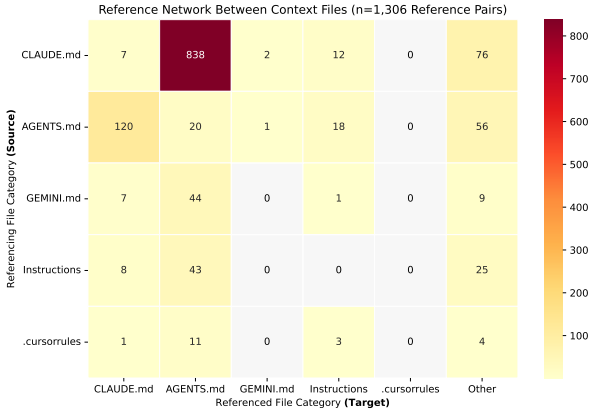


Figure 7: Reference network between context file types. Each cell shows how many files of the source type (row) reference a file of the target type (column).

illustrating the rapid growth of `CLAUDE.md` and `AGENTS.md` from late 2024 and the more recent emergence of skills and subagents.

Configuration practices. The raw configuration files in `repos_data.7z` allow qualitative and quantitative content analysis. What instructions do developers give AI agents? Do configuration patterns differ by programming language or project size? How do context files for different tools compare in length, structure, and content? Researchers can also compare how the eight configuration mechanisms are combined within projects: our companion study [6] found that most repositories use only context files, while advanced mechanisms such as skills and subagents remain rare.

The Context file metadata records whether a file is a reference to another file (e.g., a `CLAUDE.md` that simply points to a shared `AGENTS.md`). Figure 7 shows the resulting reference network: `AGENTS.md` is the dominant reference target, receiving incoming references from tool-specific context files of all five tools. This pattern suggests that `AGENTS.md` is emerging as a shared, vendor-neutral configuration layer that tool-specific files delegate to.

AI-co-authored commits. The `commits.csv` file enables analysis of AI-co-authored commits: their frequency, timing, commit

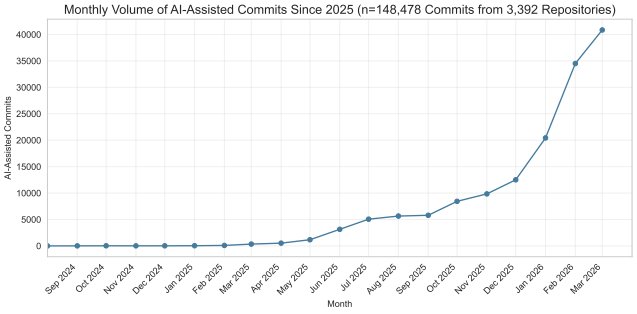


Figure 8: Monthly volume of AI-co-authored commits across the 4,741 repositories with detected configuration artifacts.

message style, and distribution across projects. Figure 8 shows the corresponding acceleration in AI-co-authored commits, with monthly volumes growing from near zero in early 2024 to over 40,000 by March 2026. The sharp uptick coincides with the availability of agentic features in Claude Code and GitHub Copilot, and suggests that configuration artifact adoption and active AI tool usage are closely linked.

Each commit record includes the full commit message and the detected AI tool, allowing researchers to compare the writing style and scope of AI-generated commits with human-authored ones. Researchers can also measure what fraction of a project's total commits are AI-co-authored, and how this fraction changes over time.

Repository characteristics. Because `repos.csv` covers both tool-adopting and non-adopting repositories with full GitHub metadata, researchers can study what distinguishes projects that adopt AI coding tools, using metrics such as project size, contributor count, language, and activity level. The classification justifications in `repos.csv` can also serve as training data for automated repository classification approaches.

Governance and safety. The raw configuration files encode how developers constrain AI agent behavior: which files agents may modify, which actions require human approval, and what security or coding practices agents must follow. Researchers can study how governance and safety constraints are expressed across projects, identify under-specified areas, and detect risks such as ambiguity, over-delegation, or missing safeguards. Repositories that configure multiple tools simultaneously allow studying whether instructions across artifacts are consistent or contradictory.

Model training and evaluation. The 45,126 configuration files provide a corpus for training or fine-tuning language models on developer instructions for AI agents, a text genre that blends documentation, specification, and prompt engineering. Because the dataset links configuration artifacts to AI authorship metadata, researchers can filter for human-authored configurations when building training sets, avoiding circular effects from training on AI-generated content. The per-artifact metadata (creation date, commit count, language) enables stratified sampling for controlled experiments.

6 Reproducibility and Availability

The dataset is archived on Zenodo under CC BY 4.0 [1]. It includes all CSV files listed in Table 1 and the `repos_data.7z` archive containing the raw text of all 45,126 collected configuration files. Large files (`repos.csv` and `commits.csv`) are additionally distributed as 7z archives for faster download. Users can load the data directly with standard tools and join across files:

```
import pandas as pd
repos = pd.read_csv("repos.csv")
configured = repos[repos["scanned_at"].notna()]
claude_repos = configured[configured["claude"] ==
    True]
commits = pd.read_csv("commits.csv")
claude_commits = commits[commits["ai_tool"].str.
    contains("Claude")]
```

The complete construction pipeline (all Python scripts and YAML configuration files across the three stages shown in Figure 5) is archived separately on Zenodo [2]. The pipeline is organized into four directories matching the methodology sections: `1_sampling/`, `2_classification/`, `3_data-collection/`, and `4_data-analysis/`. Each stage reads from the previous stage's output and produces self-contained results. The sampling stage is driven by a declarative YAML configuration file that specifies all filter parameters, making queries reproducible given the same GHS snapshot. The classification stage depends on the OpenAI API (GPT-5.2 and GPT-5-nano), introducing potential non-determinism across runs; we include the classification justifications in the dataset so that users can inspect and verify individual labels. The artifact extraction stage uses deterministic file-matching heuristics documented in a versioned specification, with unit tests for the AI commit detection logic.

An interactive website[2] provides a searchable interface for browsing repositories, viewing configuration artifacts, exploring AI-co-authored commits, and consulting usage examples without downloading the full dataset.

7 Limitations

As with any large-scale mined dataset, this dataset has limitations stemming from sampling decisions, automated classification, and heuristic-based extraction.

The dataset covers only public GitHub repositories and does not include proprietary codebases, where agentic AI coding tools may be adopted at a larger scale and under different organizational constraints. AI tool adoption on other platforms (GitLab, Bitbucket) may follow different patterns; extending the pipeline to other platforms would require adapting the repository sampling and cloning stages.

LLM-based classification of engineered projects introduces non-determinism: re-running GPT-5.2 on the same input may produce slightly different labels. We include justification texts in the dataset, and two authors spot-checked labels on a random sample. The 152 unsure repositories are included in `repos.csv` with their labels, so users can apply their own inclusion criteria.

[2]https://se-uhd.de/ai-config-dataset/

Tool detection relies on file and directory naming conventions documented by each tool's vendor. Repositories that use non-standard locations or that removed configuration files before our March 2026 snapshot are missed. The detection heuristics are versioned in the pipeline archive and can be extended as tools evolve.

AI-co-authored commit detection depends on identifiable markers in author fields, git trailers, and commit messages. Tools or workflows that do not leave such traces produce false negatives, meaning the true volume of AI-assisted contributions is likely underestimated. Some tools do not expose consistent attribution signals, which points to the need for more systematic approaches to identify AI-authored commits regardless of whether explicit attribution is present. The detection patterns are documented and extensible.

The dataset reflects a single point-in-time snapshot (March 2026). We designed the pipeline for periodic re-execution with updated data.

8 Conclusion

We presented a curated dataset of agentic AI coding tool configurations from 4,741 open-source GitHub repositories, covering five tools, eight configuration mechanisms, 15,612 configuration artifacts with their full content, and 148,551 AI-co-authored commits. The dataset and the construction pipeline are publicly available on Zenodo. An interactive website provides detailed usage examples and an alternative entry point for browsing and exploring the dataset without downloading it.

We intend to continuously update the dataset as new tools and configuration mechanisms emerge, expanding the detection heuristics accordingly. Future versions will also improve the classification pipeline by incorporating quantitative repository metadata (stars, forks, activity levels) and the GitHub repository description alongside the README content, which should further reduce the number of unsure cases. Beyond maintenance, future work can link configuration practices to downstream outcomes such as code quality and review efficiency.

References

[1] Sebastian Baltes, Seyedmoein Mohsenimofidi, Levi Böhme, Jai Lal Lulla, Muhammad Auwal Abubakar, Christoph Treude, and Matthias Galster. 2026. A Dataset of Agentic AI Coding Tool Configurations. doi:10.5281/zenodo.19375880

[2] Sebastian Baltes, Seyedmoein Mohsenimofidi, Levi Böhme, Jai Lal Lulla, Muhammad Auwal Abubakar, Christoph Treude, and Matthias Galster. 2026. A Dataset of Agentic AI Coding Tool Configurations (Pipeline). doi:10.5281/zenodo.19375429

[3] Worawalan Chatlatanagulchai, Hao Li, Yutaro Kashiwa, Brittany Reid, Kundjanasith Thonglek, Pattara Leelaprute, Arnon Rungsawang, Bundit Manaskasemsak, Bram Adams, Ahmed E. Hassan, and Hajimu Iida. 2025. Agent READMEs: An Empirical Study of Context Files for Agentic Coding. *CoRR* abs/2511.12884 (2025). arXiv:2511.12884 doi:10.48550/ARXIV.2511.12884

[4] Ozren Dabic, Emad Aghajani, and Gabriele Bavota. 2021. Sampling Projects in GitHub for MSR Studies. In *18th IEEE/ACM International Conference on Mining Software Repositories, MSR 2021, Madrid, Spain, May 17-19, 2021*. IEEE, 560–564. doi:10.1109/MSR52588.2021.00074

[5] Yihong Dong, Xue Jiang, Zhi Jin, and Ge Li. 2024. Self-Collaboration Code Generation via ChatGPT. *ACM Trans. Softw. Eng. Methodol.* 33, 7 (2024), 189:1–189:38. doi:10.1145/3672459

[6] Matthias Galster, Seyedmoein Mohsenimofidi, Jai Lal Lulla, Muhammad Auwal Abubakar, Christoph Treude, and Sebastian Baltes. 2026. Configuring Agentic AI Coding Tools: An Exploratory Study. In *3rd ACM International Conference on AI-powered Software (AIware 2026)*. arXiv:2602.14690 Accepted, to appear.

[7] 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. *CoRR* abs/2509.06216 (2025). arXiv:2509.06216 doi:10.48550/ARXIV.2509.06216

[8] Hao He, Courtney Miller, Shyam Agarwal, Christian Kästner, and Bogdan Vasilescu. 2026. Speed at the Cost of Quality: How Cursor AI Increases Short-Term Velocity and Long-Term Complexity in Open-Source Projects. In *23rd IEEE/ACM International Conference on Mining Software Repositories (MSR 2026)*. arXiv:2511.04427

[9] Kosei Horikawa, Hao Li, Yutaro Kashiwa, Bram Adams, Hajimu Iida, and Ahmed E. Hassan. 2025. Agentic Refactoring: An Empirical Study of AI Coding Agents. *CoRR* abs/2511.04824 (2025). arXiv:2511.04824 doi:10.48550/ARXIV.2511.04824

[10] Shaokang Jiang and Daye Nam. 2026. Beyond the Prompt: An Empirical Study of Cursor Rules. In *23rd IEEE/ACM International Conference on Mining Software Repositories (MSR 2026)*. arXiv:2512.18925

[11] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. Germán, and Daniela E. Damian. 2014. The promises and perils of mining GitHub. In *11th Working Conference on Mining Software Repositories, MSR 2014, Proceedings, May 31 - June 1, 2014, Hyderabad, India*, Premkumar T. Devanbu, Sung Kim, and Martin Pinzger (Eds.). ACM, 92–101. doi:10.1145/2597073.2597074

[12] 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. *CoRR* abs/2507.15003 (2025). arXiv:2507.15003 doi:10.48550/ARXIV.2507.15003

[13] Seyedmoein Mohsenimofidi, Matthias Galster, Christoph Treude, and Sebastian Baltes. 2026. Context Engineering for AI Agents in Open-Source Software. In *23rd IEEE/ACM International Conference on Mining Software Repositories (MSR 2026)*. arXiv:2510.21413

[14] Nuthan Munaiah, Steven Kroh, Craig Cabrey, and Meiyappan Nagappan. 2017. Curating GitHub for engineered software projects. *Empir. Softw. Eng.* 22, 6 (2017), 3219–3253. doi:10.1007/S10664-017-9512-6

[15] Gede Artha Azriadi Prana, Christoph Treude, Ferdian Thung, Thushari Atapattu, and David Lo. 2019. Categorizing the Content of GitHub README Files. *Empir. Softw. Eng.* 24, 3 (2019), 1296–1327. doi:10.1007/S10664-018-9660-3

[16] Romain Robbes, Théo Matricon, Thomas Degueule, André C. Hora, and Stefano Zacchiroli. 2026. Agentic Much? Adoption of Coding Agents on GitHub. *CoRR* abs/2601.18341 (2026). arXiv:2601.18341 doi:10.48550/ARXIV.2601.18341

[17] Ranjan Sapkota, Konstantinos I. Roumeliotis, and Manoj Karkee. 2025. AI Agents vs. Agentic AI: A Conceptual Taxonomy, Applications and Challenges. *CoRR* abs/2505.10468 (2025). arXiv:2505.10468 doi:10.48550/ARXIV.2505.10468

[18] Stack Exchange Inc. 2025. Stack Overflow Developer Survey 2025: AI Agent out-of-the-box tools. https://survey.stackoverflow.co/2025/ai/#3-ai-agent-out-of-the-box-tools.

[19] 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. *CoRR* abs/2509.14745 (2025). arXiv:2509.14745 doi:10.48550/ARXIV.2509.14745

[20] Tao Xiao, Youmei Fan, Fabio Calefato, Christoph Treude, Raula Gaikovina Kula, Hideaki Hata, and Sebastian Baltes. 2025. Self-Admitted GenAI Usage in Open-Source Software. arXiv:2507.10422 [cs.SE]

[21] Yangtian Zi, Zixuan Wu, Aleksander Boruch-Gruszecki, Jonathan Bell, and Arjun Guha. 2025. AgentPack: A Dataset of Code Changes, Co-Authored by Agents and Humans. arXiv:2509.21891 [cs.SE]