

# Configuring Agentic AI Coding Tools: An Exploratory Study

Matthias Galster  
University of Bamberg  
Germany  
mgalster@ieee.org

Syedmoein Mohsenimofidi  
Heidelberg University  
Germany  
s.mohsenimofidi@uni-heidelberg.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

Agentic AI coding tools increasingly automate software development tasks. Developers can configure these tools through versioned repository-level artifacts such as Markdown and JSON files. We present a systematic analysis of configuration mechanisms for agentic AI coding tools, covering Claude Code, GitHub Copilot, Cursor, Gemini, and Codex. We identify eight configuration mechanisms spanning a spectrum from static context to executable and external integrations, and, in an empirical study of 2,923 GitHub repositories, examine whether and how they are adopted, with a detailed analysis of CONTEXT FILES, SKILLS, and SUBAGENTS. First, CONTEXT FILES dominate the configuration landscape and are often the sole mechanism in a repository, with AGENTS.md emerging as an interoperable standard across tools. Second, advanced mechanisms such as SKILLS and SUBAGENTS are only shallowly adopted. Most repositories define only one or two artifacts, and SKILLS predominantly rely on static instructions rather than executable workflows. Third, distinct configuration cultures are forming around different tools, with Claude Code users employing the broadest range of mechanisms. These findings establish an empirical baseline for understanding how developers configure agentic tools, suggest that AGENTS.md serves as a natural starting point, and motivate longitudinal and experimental research on how configuration strategies evolve and affect agent performance.

## CCS Concepts

- Software and its engineering;

## Keywords

Software Engineering, Generative AI, AI Agents, Configuration

## ACM Reference Format:

Matthias Galster, Syedmoein 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)*, July 6–7, 2026, Montreal, Canada.



This work is licensed under a Creative Commons Attribution 4.0 International License. *AIware 2026, Montreal, Canada*  
© 2026 Copyright held by the owner/author(s).  
ACM ISBN 979-8-4007-XXXX-X/2026/04  
<https://doi.org/10.1145/XXXXXXX.XXXXXXX>

Montreal, Canada. ACM, New York, NY, USA, 9 pages. <https://doi.org/10.1145/XXXXXXX.XXXXXXX>

## 1 Introduction

Agentic AI coding tools based on large language models (LLMs) [25, 27], automate time-consuming and repetitive tasks, such as generating and editing code, tests, and documentation. Unlike purely reactive conversational assistants, these tools proactively accomplish defined objectives [3] by autonomously interacting with development environments, project artifacts, external data, and command-line tools with reduced human interaction [29]. Certain functionality can be delegated to *tools* and (*sub*)*agents*. *Tools* are deterministic capabilities with a specific bounded function (e.g., `run_python(code)` or `search_web(query)`), invoked by the underlying model. We distinguish such fine-grained tools from full-fledged *agentic tools* (e.g., Claude Code). (*Sub*)*agents* are goal-directed components that interpret a user goal, decompose it into substeps, select and execute tools, and iteratively adjust their plan in an “agent loop.” Initially, agentic AI coding tools such as Claude Code and OpenAI Codex implemented one central agent loop steered by a foundation model, followed by conversational tools such as GitHub Copilot and Cursor offering similar capabilities in an “agent mode.” More recently, tool vendors introduced extension and configuration mechanisms that allow developers to write their own (sub)agents that operate in parallel to the central agent loop.

We define a *configuration mechanism* as a means for developers to tailor tool and agent behavior to a project or workflow (e.g., context files or dedicated subagents). A *configuration artifact* is a tangible instance of a mechanism (e.g., the actual Markdown file). For example, files like AGENTS.md or CLAUDE.md act as “READMEs for agents” with context-specific information about build commands, coding conventions, and rules for CI/CD pipelines [17, 19]. Configuration artifacts are version-controlled, inspectable, and collaboratively maintained. With the increased availability and diversity of agentic tools, the number of configuration mechanisms and related artifacts has also increased.

To understand the current landscape of agentic tools’ configuration mechanisms and their use in open-source software (OSS), we address the following **research questions**:

- RQ1** What configuration mechanisms do agentic coding tools offer?
- RQ2** Which mechanisms are adopted in OSS repositories?
- RQ3** How are configuration mechanisms adopted?

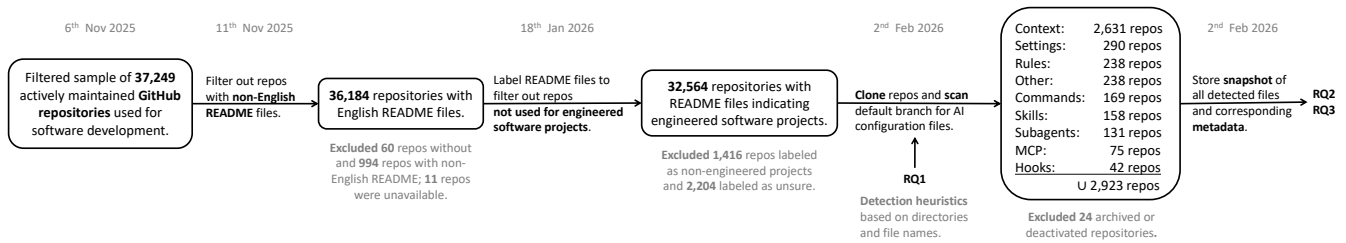


Figure 1: Data collection process.

We consider configuration mechanisms that are captured in configuration artifacts to be consumed by agentic tools, are repository-versioned for collaborative maintenance, and include instructions intended to customize the behavior of AI coding tools.

With this paper, we provide the following **contributions**: (1) We systematically documented eight configuration mechanisms: CONTEXT FILES, SKILLS, SUBAGENTS, COMMANDS, RULES, SETTINGS, HOOKS, and MCP servers. We identified these from the documentation of Claude Code, GitHub Copilot, Cursor CLI, Gemini CLI, and Codex CLI. Some mechanisms are available in all tools; others are specific to particular tools. (2) We analyzed the adoption of these configuration mechanisms in 2,923 GitHub repositories. CONTEXT FILES (Markdown files that provide contextual project information) dominated and are often the sole configuration mechanism. Claude Code users apply the broadest range of configuration mechanisms. (3) We analyzed the adoption of CONTEXT FILES, SKILLS (bundles of information and tools), and SUBAGENTS (specialized agents with their own model context), in more detail. Three CONTEXT FILE formats showed the most dynamic development, with AGENTS.md emerging as a standard. Although SKILLS and SUBAGENTS can be defined for a wide range of purposes, most repositories that adopt them define only one or two artifacts. Moreover, SKILLS primarily rely on static resources rather than executable scripts to extend tool and agent behavior.

## 2 Related Work

Prior work has examined how repository-level context files, prompts, and structured context mechanisms shape agent behavior, efficiency, and integration into development workflows, but has typically focused on single artifact types in isolation. There has been no systematic examination of the broader configuration mechanisms to customize agentic behavior. Context engineering builds on earlier prompt engineering techniques [9, 21] but broadens the scope from individual prompts to the systematic design of all information supplied to a tool. Mei et al. [16] survey this emerging field, while practitioner accounts demonstrate how structuring project-specific context improves agent effectiveness in complex codebases [12, 23]. Empirical work has mainly focused on repository-maintained context files, such as AGENTS.md. Mohsenimofidi et al. [17] analyzed the emergence of context engineering practices in open-source projects and showed that repository-level context files serve as persistent configuration mechanisms that encode architectural constraints, build commands, and workflow conventions. Chatlatanagulchai et al. [6] found that context files are actively maintained through frequent, small changes and focus predominantly on development

instructions such as build and testing procedures. Santos et al. [22] analyzed 328 CLAUDE.md files and found that architectural concerns dominate, often co-occurring with development guidelines, testing, and project overview.

Beyond structural characterization, Lulla et al. [14] conducted a controlled study comparing agent executions with and without an AGENTS.md file and reported lower runtime and token consumption while maintaining comparable task completion behavior.

Villamizar et al. [28] argued that prompts should be treated as software engineering artifacts and outlined a research agenda for their evolution, reuse, and governance. Q. Zhang et al. [31] proposed a framework treating contexts as evolving instructions, addressing brevity bias and context collapse through incremental updates.

Other work formalizes context as a programmable abstraction. Y. Zhang et al. [32] introduced Monadic Context Engineering, modeling context construction and transformation through monadic composition. McMillan [15] evaluated structured file-native context schemas and showed how format and multi-file organization influence retrieval accuracy and downstream agent performance. Ye et al. [30] proposed Meta Context Engineering, a framework in which agentic skills and context artifacts co-evolve via evolutionary search. Jiang and Nam [13] explored Cursor rules and developed a taxonomy of project contexts.

However, no prior work maps configuration mechanisms across tools or studies their adoption patterns across repositories. We provide this cross-tool perspective.

## 3 Data Collection and Analysis

We collected OSS projects hosted on GitHub, selecting repositories belonging to “engineered” software projects [18] using an updated version of a selection approach from prior work, starting with the SEART GitHub search tool [8, 17, 24]. To ensure sufficient project maturity and development history, we selected non-fork repositories with at least two contributors and a license that were created before 1 January 2024 with commits since 1 June 2025. We then excluded archived, disabled, or locked repositories. We applied a licensing filter and a language filter based on popularity, selecting Python, TypeScript, JavaScript, Go, Java, C++, Rust, PHP, C#, and C, and excluded repositories with fewer than 271 commits or fewer than 7 watchers (median-based thresholds from prior research [17]). This resulted in a sample of 37,249 repositories.

Figure 1 outlines the data collection pipeline. We cloned the repositories and searched their default branch for a README file, excluding 11 that became unavailable and 60 without a README. We then used the `lingua-language-detector` Python library to

**Table 1: Overview of repository-level configuration mechanisms across agentic AI coding tools. Each cell lists the repository file(s) or directory implementing that mechanism; “–” indicates that the mechanism is not available.**

Mechanism	Description	Claude	Copilot	Codex	Cursor	Gemini
CONTEXT FILES	Markdown file providing persistent context loaded every session.	CLAUDE.md	.github/ { copilot-instructions.md   instructions/*.md } <sup>a</sup>	AGENTS.md, AGENTS.override.md	AGENTS.md, .cursorsrules <sup>c</sup>	GEMINI.md
SETTINGS	JSON/TOML config for project-level tool behavior.	.claude/ settings (local)?.json	– <sup>b</sup>	.codex/ config.toml	.cursor/ cli.json	.gemini/{ settings.json   config.yaml }
SKILLS	Reusable knowledge and invocable workflows.	.claude/skills/	.github/skills/	.codex/skills/	.cursor/skills/	.gemini/skills/
SUBAGENTS	Specialized agents running in an isolated context.	.claude/agents/	.github/agents/	–	.cursor/agents/	–
COMMANDS	User-triggered shortcuts for predefined prompts.	.claude/ commands/	–	–	.cursor/ commands/	.gemini/ commands/
HOOKS	Scripts executed at specific agent lifecycle points.	.claude/ settings.json	.github/hooks/*.json	–	.cursor/ hooks.json	.gemini/ settings.json
RULES	System-level instructions to control agent behavior.	–	–	.codex/rules/	.cursor/rules/	–
MCP	External tool or data connections via the Model Context Protocol.	.mcp.json	– <sup>b</sup>	.codex/ config.toml	.cursor/ mcp.json	.gemini/ settings.json

<sup>a</sup> Copilot also supports CLAUDE.md, AGENTS.md, and GEMINI.md; <sup>b</sup> Configured via the GitHub web UI, not via files in the project repository;

<sup>c</sup> Cursor deprecated .cursorsrules and now suggests using AGENTS.md instead.

**Table 2: Release dates of agentic AI coding tools and repositories ( $n = 2,923$ ; one repository can use multiple tools).**

Agentic Tool	Release (Month/Year)	#Repositories
Claude Code	02/2025 (CLI & agents since release)	1,305
GitHub Copilot	10/2021 (Release)	957
	02/2025 (Copilot Agent Mode) 09/2025 (Copilot CLI)	
AGENTS.md <sup>d</sup>	05/2025 (Adoption in Codex) 08/2025 (Specification [2])	528
Cursor CLI	03/2023 (Release)	327
	06/2025 (Cursor Agents) 08/2025 (Cursor CLI)	
Gemini CLI	02/2024 (Release)	178
	05/2025 (Gemini Agent Mode)	
	06/2025 (Gemini CLI)	
Codex CLI	04/2025 (CLI & agents since release)	4

<sup>d</sup> Repositories using AGENTS.md without any tool-specific configuration artifact.

detect each file’s language, excluding 994 repositories with non-English README files.

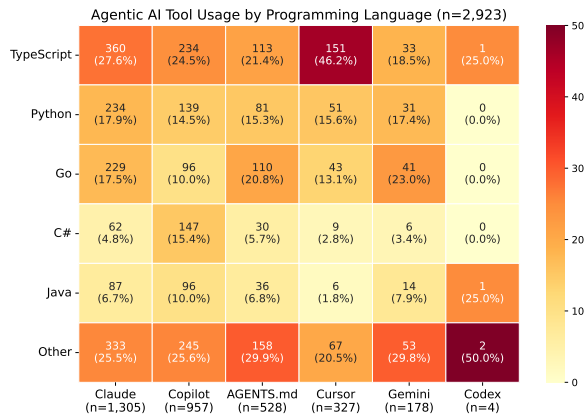
For the remaining 36,184 repositories, we used a GPT-5.2-based classification pipeline to determine, based on README content, whether each project qualifies as “engineered” per Munaiah et al. [18], that is, showing clear evidence of software engineering practices (e.g., a stated purpose, installation and deployment instructions, documentation, testing, CI/CD) and maintenance. The supplementary material includes the prompt (iteratively developed and tested on subsets) and model configuration (OpenAI defaults) [10]. The pipeline labeled 32,564 repositories as engineered and excluded

**Table 3: Repository metadata by agentic tool. Cells show median, IQR, and Cliff’s  $\delta$  for tool-adopting repositories, each compared against the complement of non-adopting repositories. Significant differences (BH-adjusted) are in bold.**

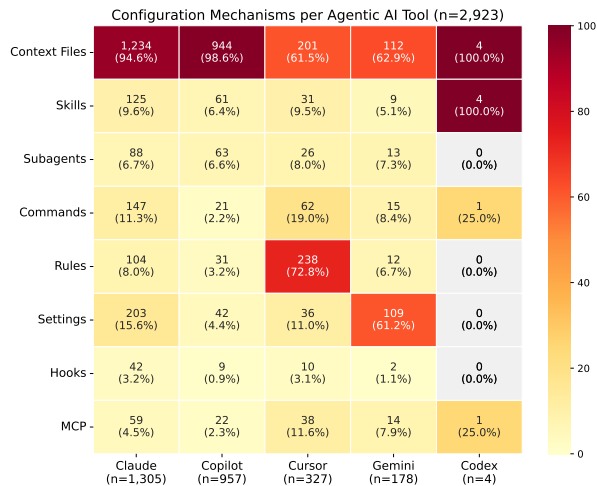
Agentic tool	Age (years)	Contrib.	Commits	Size (KB)
All ( $n=2,923$ )	6.7 (4.3–9.4)	42 (19–105)	2,108 (965–5,065)	39k (9,501–132k)
Claude ( $n=1,305$ )	<b>6.2***</b> ( <b>4.1–9.2</b> )	40 (19–111)	2,217 (989–5,270)	42k (9,979–152k)
	$\delta=-0.08$	$\delta=-0.00$	$\delta=0.03$	$\delta=0.04$
Copilot ( $n=957$ )	<b>7.1***</b> ( <b>5.0–9.7</b> )	42 (19–113)	2,231 (1,001–5,607)	45k (9,958–143k)
	$\delta=0.11$	$\delta=0.03$	$\delta=0.04$	$\delta=0.04$
AGENTS.md ( $n=528$ )	7.1 (4.4–9.4)	44 (19–89)	<b>1,809***</b> ( <b>869–3,972</b> )	<b>31k***</b> ( <b>8,511–91k</b> )
	$\delta=0.03$	$\delta=-0.02$	$\delta=-0.10$	$\delta=-0.10$
Cursor ( $n=327$ )	<b>5.5***</b> ( <b>3.4–7.9</b> )	<b>52*</b> ( <b>23–118</b> )	<b>2,780***</b> ( <b>1,264–6,029</b> )	<b>75k***</b> ( <b>21k–198k</b> )
	$\delta=-0.20$	$\delta=0.09$	$\delta=0.14$	$\delta=0.22$
Gemini ( $n=178$ )	6.8 (4.7–9.5)	<b>63***</b> ( <b>31–142</b> )	<b>3,302***</b> ( <b>1,282–9,711</b> )	<b>58k*</b> ( <b>15k–189k</b> )
	$\delta=0.02$	$\delta=0.20$	$\delta=0.20$	$\delta=0.12$
Codex ( $n=4$ )	7.6 (4.6–11)	46 (32–62)	6,014 (4,997–6,514)	<b>505k*</b> ( <b>153k–939k</b> )
	$\delta=0.12$	$\delta=-0.07$	$\delta=0.48$	$\delta=0.66$

Note: Mann–Whitney U test (two-sided) comparing each tool’s adopters against non-adopters (complement), with Benjamini–Hochberg FDR correction (6 categories  $\times$  4 metrics = 24 comparisons). Effect sizes: Cliff’s delta. \*  $p < .05$ ; \*\*  $p < .01$ ; \*\*\*  $p < .001$  (adjusted). Interpret Codex results ( $n=4$ ) with caution due to sample size.

the remaining 3,620 (1,416 classified as non-engineered, 2,204 “unsure”). We spot-checked randomly selected samples from each category during prompt development and for the final sample. We then cloned the remaining 32,564 repositories and applied heuristics



**Figure 2: Adoption of agentic tools per programming language; AGENTS.md denotes repositories using only that file with no tool-specific configuration. Percentages relative to repository count per tool. Column totals can exceed the overall repository count (multiple tools per repository).**



**Figure 3: Usage of configuration mechanisms across agentic tools. Repositories using only AGENTS.md without tool-specific configuration are excluded. Percentages relative to repository count per tool. Column totals can exceed the overall repository count (multiple tools per repository).**

based on file names and file paths to detect the usage of AI coding tools and configuration mechanisms (see Table 1). These heuristics are based on the answers to **RQ1** and are briefly discussed in Section 4. This resulted in 2,923 repositories that use one or more AI coding tools (three additional repositories contained only empty configuration artifacts and were excluded). Our data collection and analysis scripts and the analyzed data are available online [10].

## 4 Configuration Mechanisms (RQ1)

We selected five agentic AI coding tools (Table 2) based on the *2025 Stack Overflow Developer Survey* [26]: the four most popular AI tools among the surveyed developers, substituting Codex for ChatGPT as the agentic tool by the same vendor, plus Cursor, which has recently been studied in software engineering [11, 13]. Our study focuses on the CLI-based agentic interfaces that first appeared in 2025; for some tools, prior non-agentic versions exist. One author systematically reviewed each tool’s online documentation, documenting configuration mechanisms together with repository-level files and directories that indicate their usage. This was then cross-checked by two other authors and discussed among all three.

We used these findings to develop heuristics for detecting tool usage and configuration mechanisms in repositories (Table 1). For example, a `.claude` directory or a `CLAUDE.md` file indicates Claude Code usage; a `.claude/agents/` directory with Markdown files indicates SUBAGENTS. AGENTS.md is a special case: as a tool-agnostic standard, it is supported by multiple tools, so repositories using only AGENTS.md without tool-specific artifacts are tracked separately (see Table 2, footnote <sup>d</sup>). Note that for Copilot, Cursor, and Gemini, certain detected files apply to both conversational and agentic interfaces. All documents and the Python scripts that implement our matching strategy are part of the supplementary material [10].

### RQ1 (Summary):

- We identified eight configuration mechanisms spanning a spectrum from static context (e.g., CONTEXT FILES) to executable and external integrations (e.g., SKILLS, MCP). Two mechanisms (CONTEXT FILES and SKILLS) are supported by all five tools.
- Despite this convergence, no single tool implements all eight mechanisms.

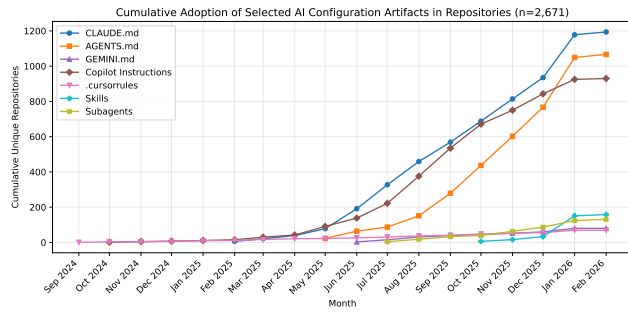
## 5 Adoption of Configuration Mechanisms (RQ2)

To understand how developers adopt configuration mechanisms in agentic AI coding tools, we analyze the presence and co-occurrence of configuration artifacts across repositories.

### 5.1 Characterization of Repositories in Dataset

Before analyzing adoption, we characterize the 2,923 repositories in our dataset to contextualize the subsequent analysis.

Of the 2,923 repositories, 2,018 (69.0%) adopted a single tool, while 299 (10.2%) configured exactly two tools and 50 (1.7%) configured three or more. An additional 528 repositories (18.1%) use AGENTS.md as a tool-agnostic standard without any tool-specific configuration artifact; these are shown separately in the analysis as “AGENTS.md” (see Table 2, footnote <sup>d</sup>). Among repositories with multiple tools, Claude appeared most frequently with others. The most common combination was Claude and Copilot ( $n = 168$ ), followed by Claude and Cursor ( $n = 145$ ), Claude and Gemini ( $n = 54$ ), and Copilot and Cursor ( $n = 49$ ). Notably, 44.3% of Cursor repositories also configure Claude. The dominance of single-tool repositories limits the potential for multi-tool confounding in tool-specific analyses. In our initial sample, the top five main programming



**Figure 4: Cumulative adoption of selected configuration artifacts for agentic tools. Copilot Instructions comprise two CONTEXT FILE artifact types (see Table 1).**

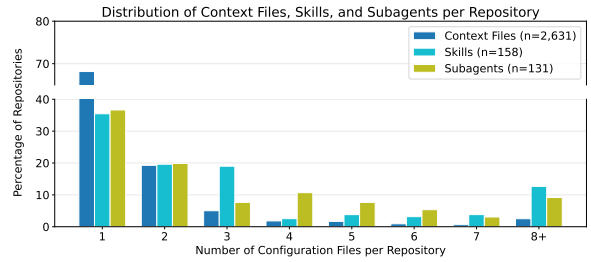
languages per repository ( $n = 36, 184$ ) were Python (8,133; 22.5%); TypeScript (4,999; 13.8%); Java (3,813; 10.5%); Go (3,786; 10.5%); and JavaScript (3,709; 10.3%). Interestingly, this order is slightly different for repositories that use agentic coding tools ( $n = 2, 923$ , see Figure 2) where TypeScript and Go were more prominent and C# replaces JavaScript in the top five: TypeScript (745, 25.5%); Python (485, 16.6%); Go (457, 15.6%); C# (228, 7.8%); Java (227, 7.8%). TypeScript is the most common primary language for repositories using Claude, Copilot, and Cursor. Usage in Java and C# repositories is lower for all tools but Copilot. As shown in Table 3, which compares each tool’s adopting repositories against the complement of non-adopting repositories, repositories using Cursor are younger compared to those using other tools. Excluding Codex ( $n=4$ ), repositories associated with Gemini exhibit larger contributor counts and commit volumes, compared to those associated with other tools. Cursor repositories are the largest in terms of kilobytes of source code (excluding Codex). AGENTS.md repositories (those using only AGENTS.md with no tool-specific configuration) tend to have fewer commits and smaller size than the overall population, suggesting that these repositories adopted AGENTS.md as a lightweight, tool-agnostic starting point.

## 5.2 Distribution of Configuration Mechanisms

Figure 3 shows the distribution of configuration mechanisms per tool. CONTEXT FILES (e.g., CLAUDE.md, AGENTS.md) were the most frequently adopted mechanism, used by 61.5–100% of repositories across all tools. Beyond CONTEXT FILES, distinct tool-specific profiles emerge: 72.8% of Cursor repositories adopt RULES, since Cursor was one of the first to introduce this mechanism [13], and 61.2% of Gemini repositories use SETTINGS. Other mechanisms remain below 20% of repositories for most tools.

Most repositories included only a single CONTEXT FILE artifact, and the adoption of multiple non-context file mechanisms within the same repository remained rare. These results suggest that configuration usage is characterized by one dominant baseline configuration (i.e., CONTEXT FILES).

**5.2.1 Co-occurrence of Configuration Mechanisms.** We quantified pairwise associations between mechanism adoption using



**Figure 5: Configuration mechanism count per repository.**

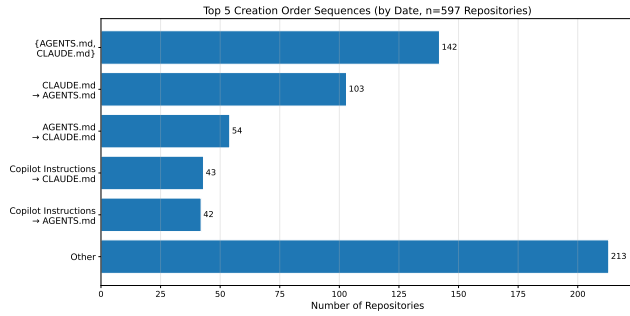
Cramér’s  $V$  from chi-squared tests (28 comparisons, Benjamini-Hochberg corrected). Several configuration mechanisms are frequently adopted together, and some of these associations can be explained by shared configuration artifacts. For example, SETTINGS and HOOKS show a positive association ( $V = 0.36$ ), because hooks are usually defined in SETTINGS files. SETTINGS and HOOKS also both co-occur with SKILLS ( $V = 0.15$  and  $V = 0.24$ , respectively), which are defined and configured differently. SUBAGENTS also show positive associations with multiple mechanisms, including SETTINGS, SKILLS, and MCP. Other configuration mechanisms appear less frequently in combination. CONTEXT FILES are negatively associated with RULES ( $V = 0.41$ ) and COMMANDS ( $V = 0.07$ ), which can be attributed to how they are defined and which tools first introduced the mechanisms. Overall, configuration mechanism adoption is characterized by recurring combinations rather than uniform usage, shaped by the tools and mechanisms they support (e.g., Claude, the dominating tool, does not support RULES). The full co-occurrence matrix is available in the supplementary material.

**5.2.2 Adoption of Configuration Mechanisms over Time.** Figure 4 shows the adoption over time of individual CONTEXT FILE artifacts, plus SKILLS and SUBAGENTS.

CONTEXT FILES (e.g., CLAUDE.md, AGENTS.md) clearly dominated and increased continuously, while SKILLS and SUBAGENTS experienced comparatively slow growth. .cursorrules and copilot-instructions.md started being introduced in 2024, although Cursor and Copilot were originally released in 2023 and 2021, respectively; their agentic capabilities were only introduced in 2025 (see Table 2). The adoption of copilot-instructions.md has increased since then.

### RQ2 (Summary):

- CONTEXT FILES are the dominant and often sole configuration mechanism; other mechanisms are adopted less frequently. Over time, the gap has widened, with SKILLS and SUBAGENTS experiencing comparatively slow growth.
- Adoption varies: Claude has a broad configuration footprint, while Cursor emphasizes rule-based mechanisms. Co-occurrence patterns are shaped by tool-specific mechanism support.
- Most repositories (69.0%) adopt a single tool, and multi-tool overlap is limited. An additional 18.1% use AGENTS.md without any tool-specific configuration.



**Figure 6: Creation order of CONTEXT FILES per repository.** Curly braces indicate that files were added on the same day.

## 6 Details of Configuration Mechanisms (RQ3)

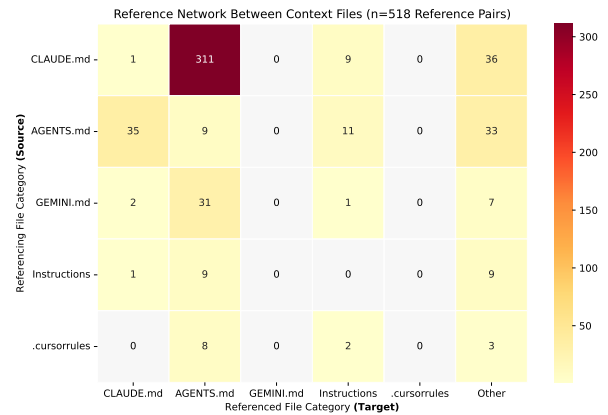
We analyzed three mechanisms in detail: CONTEXT FILES and SKILLS because they are supported by all tools, and SUBAGENTS because they follow a similar format as SKILLS.

### 6.1 Configuration Mechanism: CONTEXT FILES

CONTEXT FILES are Markdown files that provide central, machine-readable context about a project. AGENTS.md, introduced by OpenAI, now serves as an open tool-agnostic convention increasingly supported by various tools [17]. We identified 4,854 CONTEXT FILES across 2,631 of the 2,923 repositories in our sample. Most repositories include one or two such files (Figure 5).

CLAUDE.md emerges as the dominant file type with 1,658 (34.2%), followed closely by AGENTS.md and copilot-instructions.md with 1,571 (32.4%) and 1,393 (28.7%) files, respectively. GEMINI.md (159 files, 3.3%) and .cursorrules (73 files, 1.5%) are rare. Note that .cursorrules are now deprecated; Cursor suggests using AGENTS.md instead. Cursor RULES are different and are stored in .cursor/rules/. We also investigated the repository-level adoption of the CONTEXT FILES: among the 2,631 repositories with CONTEXT FILES, CLAUDE.md demonstrates the highest adoption rate with 45.4% of repositories (1,195 repos), AGENTS.md follows closely with 40.6% adoption (1,068 repos), and copilot-instructions.md appears in 35.4% of repositories (930 repos). Examining language-specific patterns, CONTEXT FILES appeared most frequently in TypeScript (651 repos), Python (435), Go (414), C# (210), Java (209), JavaScript (200), Rust (180), C++ (156), PHP (123), and C (53) repositories. CLAUDE.md was the most common CONTEXT FILE across languages, except for Java, C#, and C++, where copilot-instructions.md dominated.

Figure 6 shows the creation order of CONTEXT FILES in repositories with multiple types (single-type repositories are omitted). CLAUDE.md was typically created first, with AGENTS.md commonly added afterward—likely because Claude Code is the most popular agentic tool but does not yet support the emerging standard AGENTS.md [5]. Repositories that started with copilot-instructions.md often later added CLAUDE.md or AGENTS.md, despite Copilot supporting all major CONTEXT FILE types, reinforcing their de facto standard status.



**Figure 7: Reference-only CONTEXT FILES.** Examples of other files include README.md and CONTRIBUTING.md.

Finally, CONTEXT FILES commonly reference each other (Figure 7). We identified three types of references, where a source file points to a target file rather than providing its own content:

- (1) Direct pointer: CONTEXT FILES include one line that references files either via their filename (e.g., “AGENTS.md”, “@./AGENTS.md”), imperative statements (e.g., “Read @AGENTS.md”), or a Markdown link (e.g., “[anchor text](AGENTS.md)”).
- (2) Short reference with context: CONTEXT FILES provide 2–5 lines of what to do with the referenced file, e.g., “Use instructions from AGENTS.md to guide your work.”, “Always read AGENTS.md before answering”, or a Markdown header followed by one of the above.
- (3) Brief summary plus reference: CONTEXT FILES include a header, 1–2 sentences of context, and a reference to another file, e.g., “# AI Guidelines [...] \*\*Read @AGENTS.md for comprehensive guidelines.\*\*”

We found 518 reference pairs. CLAUDE.md had the most outgoing references (357), predominantly pointing to AGENTS.md (311 times). Conversely, AGENTS.md received the most incoming references (368) and, when acting as a pointer, most frequently referenced CLAUDE.md—reflecting the symbiotic relationship also observed in the creation order.

### 6.2 Configuration Mechanism: SKILLS

SKILLS were introduced by Anthropic and now serve as an open standard for extending agentic AI tools with specialized contextual knowledge and workflows. SKILLS bundle prompts, tools, and documentation that an agent can invoke on demand. Each SKILL is a directory containing a SKILL.md file with YAML frontmatter (name, description) and instructions in the body.

We found 601 SKILLS in 158 repositories. On average, these repositories use 3.8 SKILLS ( $min = 1.0$ ;  $max = 28.0$ ;  $median = 2$ ). However, as Figure 5 shows, repositories that use SKILLS typically use fewer than three SKILLS.

As with CONTEXT FILES, TypeScript and Python dominated the language distribution of repositories that use SKILLS, mirroring their dominance across all repositories that use agentic tools (Figure 2).

The specification recommends that SKILL.md files contain fewer than 500 lines, with detailed material moved to separate files [1]. Only 29 of the 601 SKILLS (5%) exceeded this limit. To provide detailed information and instructions, SKILLS can include additional resource directories loaded by agents when required:

- (1) `scripts/` contains executable code that agents can run. Language options include Python, Bash, and JavaScript.
- (2) `references/` contains documentation that an agent can read when needed. This can include technical references, templates or structured data and other domain-specific files.
- (3) `assets/` contains static resources, such as templates (document templates, configuration templates), images (diagrams, examples) and data files (lookup tables, schemas).

We scanned all SKILLS directories for the occurrence of resource folders. Of the 601 skills analyzed, the vast majority (514, 85.5%) included no additional resources. Among those that did, the most common patterns were a `references/` directory and a `scripts/` directory (35 skills each, 5.8%). Eleven SKILLS (1.8%) combined both `scripts/` and `references/`, while an `assets/` directory appeared in only four SKILLS (0.7%). The remaining combinations, i.e., `references/` with `assets/`, and all three directories together, each occurred just once (0.2%). In summary, SKILLS mostly use “static” resources (documentation that the agent reads when needed) rather than “dynamic” resources, such as executable scripts that extend agent behavior.

### 6.3 Configuration Mechanism: SUBAGENTS

SUBAGENTS are specialized AI agents to which agentic AI coding tools or other agents can delegate tasks. They share the same YAML frontmatter structure as SKILLS but operate within their own context window and return results to the parent agent, whereas SKILLS execute within the calling agent’s context [4, 7].

We found 450 SUBAGENTS across 131 repositories. On average, these repositories use 3.44 SUBAGENTS ( $min = 1.0$ ;  $max = 17.0$ ;  $median = 2$ ), though most use fewer than three (Figure 5). Claude Code’s SUBAGENTS can have their own “memory,” i.e., a persistent directory that survives across interactions and can build up knowledge over time, e.g., to store debugging insights [4]. However, we found no repositories storing such memory files.

#### RQ3 (Summary):

- **CONTEXT FILES:** CLAUDE.md, AGENTS.md and `copilot-instructions.md` are the most common artifacts. Creation and reference patterns revealed that CLAUDE.md and AGENTS.md serve as de facto standards while AGENTS.md emerges as a unifying standard.
- **SKILLS:** Although SKILLS can be fine-grained and offer various customization features, most repositories that adopt them only define one or two and predominantly utilize static documentation rather than dynamic scripts.

- **SUBAGENTS:** Usage patterns mirror those of SKILLS, with most repositories defining only one or two. No repositories use the persistent memory feature available for Claude Code’s SUBAGENTS.

### 6.4 Threats to Validity

We organize this section following established guidelines [20].

*Construct validity:* Our heuristics (Table 1) detect the *presence* of configuration artifacts but not whether the corresponding tool is actively used. We partially mitigate this by restricting our sample to actively maintained repositories, but artifact presence remains a proxy for adoption. As a further mitigation, we scanned the complete commit history of every repository with at least one configuration file for AI-authored commits, using author/commmitter identity fields, git trailers (e.g., Co-authored-by), and commit message patterns. Among the 2,895 repositories with at least one configuration artifact, 2,079 (71.8%) also contain AI-authored commits, providing evidence that detected artifacts are associated with active tool usage rather than being dormant files. The detection scripts and full results are included in the supplementary material. Additionally, for GitHub Copilot, Cursor, and Gemini, detected files apply to both conversational and agentic modes, so we cannot isolate agentic usage. This limitation does not affect agentic-only mechanisms such as SKILLS and SUBAGENTS, nor the broader AGENTS.md trend. Although multi-tool overlap could conflate tool-specific patterns, 85.3% of tool-adopting repositories configure a single tool, limiting this concern. To verify that detected artifacts contain meaningful content, we programmatically scanned all configuration files across the eight artifact types and excluded empty files (reducing the initial 2,926 detected repositories to 2,923). One of the authors then used Claude Code (Opus 4.6 with high effort) to check all 451 files with ten or fewer lines. Of these, 396 contained substantive configuration content (e.g., coding conventions, agent definitions, tool settings), 51 were reference-only files analyzed separately in Section 6, and four were borderline cases (e.g., a bare heading or filename). The supplementary material includes the inspection script, the checked files, and their classification.

*Internal validity:* We classified repositories as “engineered” software projects based on their README content. During iterative prompt development, we tested GPT-5.2, GPT-5-nano, and GPT-5-mini, and selected GPT-5.2 because spot-checks showed it produced fewer “unsure” labels and fewer false positives and negatives. We used a single labeling run; 2,204 “unsure” cases (often due to inaccessible linked resources) were excluded. Future work should assess the reliability of this approach by adding alternative models and configurations. Detection heuristics were designed by one author and cross-checked by two others, but rapidly evolving tool ecosystems may introduce conventions not yet captured.

*External validity:* Our study covers only open-source repositories on GitHub; practices in proprietary or enterprise settings may differ. We selected repositories showing established engineering practices, but cannot claim representativeness for closed-source development, nor did we examine variation across application domains. Finally, our findings are a point-in-time snapshot (February 2026) of a fast-moving field.

## 7 Discussion

Our study offers a first cross-tool snapshot of configuration mechanisms and their artifacts across five agentic AI coding tools and 2,923 GitHub repositories. Below, we discuss the implications of our findings.

*Standardization around AGENTS.md:* Our results reveal a clear trajectory toward AGENTS.md as a unifying, tool-agnostic configuration artifact. Creation order analysis shows that CLAUDE.md typically appears first with AGENTS.md added later (Figure 6). Reference patterns confirm this. CLAUDE.md most frequently points to AGENTS.md (311 cases), which receives 368 incoming references overall, far more than any other file type. Together, these patterns suggest bottom-up convergence around AGENTS.md as a cross-tool baseline, driven by developer practice and compatibility needs rather than vendor mandate. This is further reinforced by the 528 repositories that use AGENTS.md without any tool-specific configuration, suggesting that developers adopt it as a tool-agnostic standard even before committing to a specific tool. At the same time, the layering of multiple context files within a single repository raises the risk of redundant or conflicting instructions across artifacts. For tool vendors, native support for AGENTS.md—as already provided by Cursor and Codex—may become a baseline expectation.

*Shallow adoption despite available depth:* Across all three configuration mechanisms we examined in detail, adoption remains limited. Most repositories include only one or two CONTEXT FILES (Figure 5), and the median number of SKILLS and SUBAGENTS per repository is two in both cases. Moreover, 85.5% of SKILLS do not include additional resources, and when resources are present, static documentation (references/) is equally common as executable scripts (scripts/). In practice, SKILLS, therefore, function primarily as structured text rather than executable workflow bundles. This suggests that configuration is currently used more as documentation than as automation. We also found no evidence of repositories using the persistent memory feature of Claude Code’s SUBAGENTS.

This gap likely reflects both the novelty of these mechanisms and the effort required to configure them. Developers may gravitate toward the lowest-friction mechanism (i.e., CONTEXT FILES), since defining executable SKILLS with scripts and structured resources requires more design and maintenance effort than authoring generic Markdown-based files. Future work should assess whether deeper configuration leads to measurable performance gains, extending early evidence on the impact of CONTEXT FILES [14].

*Distinct tool ecosystems:* Configuration practices vary systematically across tools: Claude Code repositories exhibit the broadest configuration footprint, Cursor projects emphasize RULES and COMMANDS, and Copilot repositories rarely extend beyond CONTEXT FILES. These differences likely reflect the configuration options each tool exposes (Table 1) and the norms emerging within their user communities. Repository characteristics also differ: Cursor repositories tend to be younger and larger, while Gemini repositories show higher activity levels (Table 3). The 528 repositories using AGENTS.md without any tool-specific configuration further reinforce its role as a tool-agnostic standard. It remains an open

question whether these tool-specific configuration cultures will converge as tool capabilities increasingly overlap, or whether distinct usage patterns will persist.

*Practical implications and future directions:* Table 1 provides a reference for developers seeking to understand available configuration options. Based on our findings, we offer the following observations for practitioners. First, CONTEXT FILES—particularly AGENTS.md—represent the lowest-friction entry point for configuring agentic tools and are already widely adopted. Second, developers who rely on multiple tools should maintain an AGENTS.md file as a shared configuration baseline, given its cross-tool support and the reference patterns we observed. When multiple context files coexist, teams may benefit from structuring them hierarchically (e.g., using tool-specific files as adapters that reference a shared core file). Third, for recurring, well-defined workflows, SKILLS offer potential for richer configuration through scripts and structured resources, but in practice SKILLS are presently used more as static documentation than as executable workflows. Finally, adopting tool-specific mechanisms (e.g., rule-based configurations) may shape workflows in tool-dependent ways.

For researchers, our findings motivate longitudinal studies to track how configuration practices evolve as tools mature, and controlled studies to assess whether SKILLS with executable resources or dedicated SUBAGENTS provide measurable benefits over CONTEXT FILES alone [14]. Given that certain projects use multiple tools (Section 5.1), research should also investigate how conflicts between overlapping instructions across files can be detected and resolved. Our data collection and analysis pipeline [10] supports such ongoing analyses.

Finally, our findings represent a point-in-time snapshot of a fast-moving field. The trends we identify—toward standardization around AGENTS.md, shallow adoption of advanced mechanisms, and tool-specific configuration cultures—are early empirical signals rather than settled findings.

## 8 Conclusions

Our study provides the first comprehensive overview of configuration mechanisms for agentic AI coding tools, identifying eight mechanisms across five tools and analyzing their adoption in 2,923 GitHub repositories. Three findings stand out. First, CONTEXT FILES dominate and are often the only mechanism present, with AGENTS.md emerging as an interoperable standard. Second, advanced mechanisms such as SKILLS and SUBAGENTS remain shallowly adopted, with SKILLS used predominantly as static documents rather than executable workflows. Third, distinct configuration cultures are forming around different tools. For practitioners, AGENTS.md is the natural starting point for configuring agentic tools, especially in multi-tool environments, and SKILLS offer untapped potential for encoding recurring workflows beyond static instructions. Tool providers should improve onboarding for SKILLS and SUBAGENTS, which few repositories use despite their ability to encapsulate executable workflows and isolated agent contexts. For researchers, controlled studies should determine whether richer configuration improves outcomes over CONTEXT FILES alone, and longitudinal research should track how these patterns evolve.

## References

- [1] agentskills.io. 2026. Agent Skills. <https://agentskills.io/>.
- [2] agentsmd community. 2025. AGENTS.md: A Simple, Open Format for Guiding Coding Agents. Website. <https://agents.md/> Accessed 2026-01-18.
- [3] Anthropic. 2025. Claude 3.7 Sonnet and Claude Code. <https://www.anthropic.com/news/claude-3-7-sonnet>.
- [4] Anthropic. 2026. Create custom subagents. <https://code.claude.com/docs/en/sub-agents>.
- [5] anthropics/claude-code on GitHub. 2026. Feature Request: Support AGENTS.md. <https://github.com/anthropics/claude-code/issues/6235>.
- [6] Worawalan Chatlatanagulchai, Hao Li, Yutaro Kashiwa, Brittany Reid, Kundjansith Thonglek, Pattara Leelaprute, Arnon Rungsawang, Budit Manaskasemsak, Bram Adams, Ahmed E. Hassan, and Hajimu Iida. 2025. Agent READMEs: An Empirical Study of Context Files for Agentic Coding. arXiv:2511.12884 [cs.SE] doi:10.48550/arXiv.2511.12884
- [7] Cursor. 2026. Subagents. <https://cursor.com/docs/context/subagents>.
- [8] 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, Madrid, Spain, 560–564. doi:10.1109/MSR52588.2021.00074
- [9] DAIR.AI Prompt Engineering Guide. 2025. Elements of a Prompt | Prompt Engineering Guide. <https://www.promptingguide.ai/introduction/elements>.
- [10] Matthias Galster, Seyedmoein Mohsenimofidi, Jai Lal Lulla, Muhammad Auwal Abubakar, Christoph Treude, and Sebastian Baltes. 2026. *Configuring Agentic AI Coding Tools: An Exploratory Study (Supplementary Material)*. doi:10.5281/zenodo.18625980
- [11] Hao He, Courtney Miller, Shyam Agarwal, Christian Kästner, and Bogdan Vasilescu. 2025. Speed at the Cost of Quality: How Cursor AI Increases Short-Term Velocity and Long-Term Complexity in Open-Source Projects. arXiv:2511.04427 [cs.SE] doi:10.48550/arXiv.2511.04427 To appear at the 23rd IEEE/ACM International Conference on Mining Software Repositories (MSR 2026), Rio de Janeiro, Brazil.
- [12] Dexter Horthy. 2025. Getting AI to Work in Complex Codebases. <https://github.com/humanlayer/advanced-context-engineering-for-coding-agents/blob/main/ace-fca.md>.
- [13] Shaokang Jiang and Daye Nam. 2025. Beyond the Prompt: An Empirical Study of Cursor Rules. arXiv:2512.18925 [cs.SE] doi:10.48550/arXiv.2512.18925 To appear at the 23rd IEEE/ACM International Conference on Mining Software Repositories (MSR 2026), Rio de Janeiro, Brazil.
- [14] Jai Lal Lulla, Seyedmoein Mohsenimofidi, Matthias Galster, Jie M. Zhang, Sebastian Baltes, and Christoph Treude. 2026. On the Impact of AGENTS.md Files on the Efficiency of AI Coding Agents. arXiv:2601.20404 [cs.SE] doi:10.48550/arXiv.2601.20404 To appear at the 1st Journal Ahead Workshop (JAWs@ICSE 2026).
- [15] Damon McMillan. 2026. Structured Context Engineering for File-Native Agentic Systems: Evaluating Schema Accuracy, Format Effectiveness, and Multi-File Navigation at Scale. arXiv:2602.05447 [cs.SE] doi:10.48550/arXiv.2602.05447
- [16] Lingrui Mei, Jiayu Yao, Yuyao Ge, Yiwei Wang, Baolong Bi, Yujun Cai, Jiazhi Liu, Mingyu Li, Zhong-Zhi Li, Duzhen Zhang, Chenlin Zhou, Jiayi Mao, Tianze Xia, Jiafeng Guo, and Shenghua Liu. 2025. A Survey of Context Engineering for Large Language Models. arXiv:2507.13334 [cs.CL] doi:10.48550/arXiv.2507.13334
- [17] Seyedmoein Mohsenimofidi, Matthias Galster, Christoph Treude, and Sebastian Baltes. 2025. Context Engineering for AI Agents in Open-Source Software. arXiv:2510.21413 [cs.SE] doi:10.48550/arXiv.2510.21413 To appear at the 23rd IEEE/ACM International Conference on Mining Software Repositories (MSR 2026), Rio de Janeiro, Brazil.
- [18] 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
- [19] OpenAI. 2025. Introducing Codex. <https://openai.com/index/introducing-codex/>.
- [20] Per Runeson and Martin Höst. 2009. Guidelines for conducting and reporting case study research in software engineering. *Empir. Softw. Eng.* 14, 2 (2009), 131–164. doi:10.1007/S10664-008-9102-8
- [21] Pranab Sahoo, Ayush Kumar Singh, Sriparna Saha, Vinija Jain, Samrat Mondal, and Aman Chadha. 2024. A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications. arXiv:2402.07927 [cs.AI] doi:10.48550/arXiv.2402.07927
- [22] Helio Victor F. Santos, Vitor Costa, João Eduardo Montandon, and Marco Túlio Valente. 2025. Decoding the Configuration of AI Coding Agents: Insights from Claude Code Projects. arXiv:2511.09268 [cs.SE] doi:10.48550/arXiv.2511.09268
- [23] Philipp Schmid. 2025. The New Skill in AI is Not Prompting, It's Context Engineering. <https://www.philtschmid.de/context-engineering>.
- [24] SEART. 2025. GitHub Search. <https://seart-ghs.si.usi.ch/>.
- [25] Agnia Sergeev, Yaroslav Golubev, Timofey Bryksin, and Iftekhar Ahmed. 2025. Using AI-based coding assistants in practice: State of affairs, perceptions, and ways forward. *Information and Software Technology* 178 (2025), 107610. doi:10.1016/j.infsof.2024.107610
- [26] Stack Exchange Inc. 2026. 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>.
- [27] Valerio Terragni, Annie Vella, Partha Roop, and Kelly Blincoc. 2025. The Future of AI-Driven Software Engineering. *ACM Trans. Softw. Eng. Methodol.* 34, 5, Article 120 (May 2025), 20 pages. doi:10.1145/3715003
- [28] Hugo Villamizar, Jannik Fischbach, Alexander Korn, Andreas Vogelsang, and Daniel Méndez. 2025. Prompts as Software Engineering Artifacts: A Research Agenda and Preliminary Findings. In *Product-Focused Software Process Improvement - 26th International Conference, PROFES 2025, Salerno, Italy, December 1-3, 2025, Proceedings (Lecture Notes in Computer Science)*. Springer, Cham, 470–478. doi:10.1007/978-3-032-12089-2\_32
- [29] John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10-15, 2024*. Curran Associates, Inc., Red Hook, NY, USA, 50528–50652. doi:10.52202/079017-1601
- [30] Haoran Ye, Xuning He, Vincent Arak, Haonan Dong, and Guojie Song. 2026. Meta Context Engineering via Agentic Skill Evolution. arXiv:2601.21557 [cs.AI] doi:10.48550/arXiv.2601.21557
- [31] Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. 2025. Agentic Context Engineering: Evolving Contexts for Self-Improving Language Models. arXiv:2510.04618 [cs.CL] doi:10.48550/arXiv.2510.04618
- [32] Yifan Zhang, Yang Yuan, Mengdi Wang, and Andrew Chi-Chih Yao. 2025. Monadic Context Engineering. arXiv:2512.22431 [cs.AI] doi:10.48550/arXiv.2512.22431