Skill Roulette: Measuring Skill Invocation Rates Across LLM Coding Agents

JIN RAE LEE, Singapore Management University, Singapore
JAI LAL LULLA, Singapore Management University, Singapore
SEBASTIAN BALTES, Heidelberg University, Germany
CHRISTOPH TREUDE, Singapore Management University, Singapore

Agent skills package procedures for large language model (LLM) coding agents. An agent selects a skill from its name and description before reading the procedure. It may complete the task without invoking a relevant skill. We investigate how reliably coding agents invoke relevant skills and which factors affect that decision. We estimate invocation rates by repeating trials with identical inputs; invocation rate is the proportion of trials in which the agent invokes the relevant skill. We test 9 non-directional hypotheses about skill metadata, competing skills, contextual instructions, and task wording and composition. Eight concern invocation; the ninth tests whether making a skill available changes the occurrence of an action that the task does not request. We use tasks from 4 software engineering categories in a synthetic repository, running each of 4 coding-agent harnesses with 2 models. The planned 20,400 trials include 60 repetitions per Baseline category and configuration, 60 more per category in Claude Code with Sonnet 5.5 that admit each category's task and skill, and 10 per experimental condition. Comparisons are specified before collection and evaluated within each model and harness configuration using Boschloo's exact test.

CCS Concepts: • **Software and its engineering → Development frameworks and environments**.

Additional Key Words and Phrases: agent skills, LLM coding agents, skill invocation, invocation rate, empirical software engineering

1 Introduction

An agent skill is a file containing a procedure that a large language model (LLM) coding agent should follow for a class of tasks. The agent first reads the skill's name and description, assesses its relevance, and decides whether to load the body [Anthropic 2026e]. The procedure can guide the agent only after the skill body loads (Figure 1).

Successful task completion does not establish whether an agent invoked a relevant skill or followed its procedure. A trace can establish whether the skill was invoked, but deciding whether it should have been invoked requires an assessment of its relevance. When a missed invocation matters, a developer can supply the procedure directly or require the agent to invoke the skill. Understanding when such omissions occur is a prerequisite for detecting and correcting them in autonomous workflows.

Authors' Contact Information: Jin Rae Lee, Singapore Management University, Singapore, Singapore; Jai Lal Lulla, Singapore Management University, Singapore, Singapore; Sebastian Baltes, Heidelberg University, Heidelberg, Germany; Christoph Treude, Singapore Management University, Singapore, Singapore.

2026. Manuscript submitted to ACM

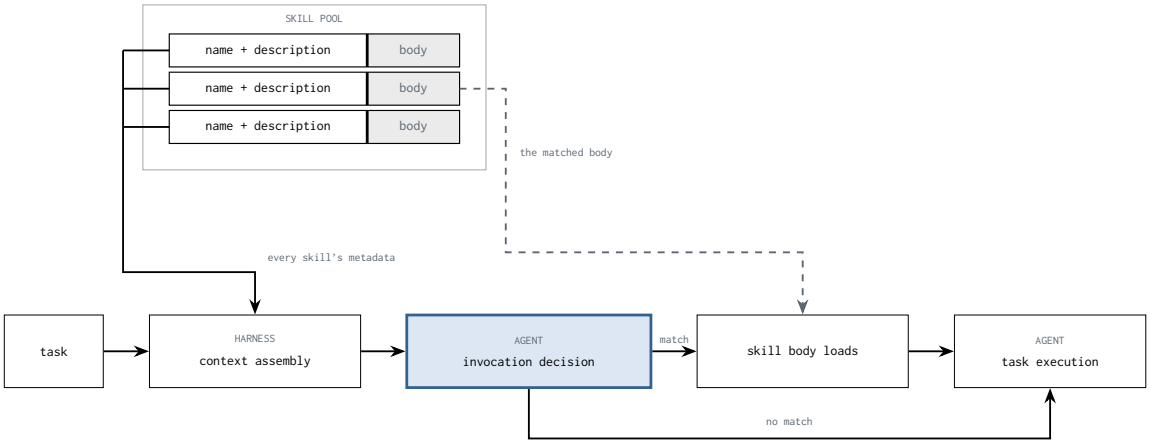


Fig. 1. *Skill invocation: a body loads only after the agent commits to the skill from name and description alone. H1–H8 measure invocation; H9 measures action occurrence.*

Invocation reliability becomes particularly important when a workflow requires several procedures. If 4 invocation decisions are independent, each has a probability of 0.6, and there is no recovery, the probability that all 4 skills are invoked is about 13%; at 0.5 each, it is about 6%. Dependencies and recovery mechanisms can change these probabilities.

This study addresses whether an available, relevant skill is invoked reliably and which changes to its metadata, task, or context affect that decision. We estimate invocation rates by repeating trials with identical inputs.

The study tests changes to skill metadata and pool composition (H1–H4), instructions and task wording (H5–H7), and the combination of tasks in a single prompt (H8). H9 asks whether making a skill available changes the occurrence of an unrequested action. We run tasks from 4 software engineering categories under 8 combinations of coding-agent harness and model. Each experimental condition has 10 repeated trials ($k = 10$); the Baseline has 60 per category and configuration to check invocation for a clearly relevant skill.

We specify the task and skill construction, experimental conditions, and comparisons needed to measure invocation separately from task completion. We will release the repository, skill inventory, and trial records for replication with other harnesses and models.

2 Background and related work

2.1 Skill configuration and discovery

Skills configure how an agent should use its capabilities on a task. Training and fine-tuning develop the model's capabilities, while tools provide the actions available to it. A skill supplies a procedure that takes effect only if the agent invokes it and follows its instructions. Skills were one of eight configuration mechanisms documented across five agentic coding tools, and one of two supported by all five [Galster et al. 2026b]. A study with a sampling frame of 40,585 repositories identified 2,430 skill artifacts in 547 repositories [Galster et al. 2026a]. Marketplace inventories can be much larger; one grew from 2,179 to 40,285 listed skills in 20 days [Ling et al. 2026]. Skills are concentrated in a few of the 18 knowledge areas in the Software Engineering Body of Knowledge (SWEBOK) [Washizaki 2024]. Gao et al. [2026b] found that Software Construction accounted for 28.3% of centralized skills and 22.1% of personal-use skills, the largest share in both populations. Construction, Operations, Testing, and Configuration Management together accounted for over 40% of centralized skills, with smaller shares in the remaining 14 areas. Metadata quality and naming overlap

motivate our experiments: 26.4% of skills sampled from public GitHub repositories had no routing description [Gao et al. 2026a], and 46.3% of listings in one marketplace shared a normalized name with another listing [Ling et al. 2026]. Shared names do not establish that procedures overlap or that near-duplicates occur together in a repository.

Gao et al. [2026a] tested whether descriptions could be compressed without changing skill selection. Of 600 skills, 252 (42.0%) preserved routing directly; restoring selected text or reverting to the original description preserved routing for all 536 eligible skills. The other 64 were excluded because their original descriptions did not trigger invocation [Gao et al. 2026a]. A separate body evaluation compared original skills, compressed skills with on-demand references, and no skills on 5 generated tasks per skill. Mean scores across 600 skills improved by 2.8% relative to the originals. These tests assessed preservation under compression and task performance, but did not report repeated identical-condition trials of invocation. We repeat each condition with the same task and pool to measure variation in invocation itself.

On SkillRouter's benchmark of 80,000 skills, hiding skill bodies reduced routing accuracy by 37–44 percentage points across dense retrieval and reranking baselines [Zheng et al. 2026]. SkillRouter retrieves and reranks candidates before presenting them to the agent, which must then decide whether to load a body from the available metadata. A coding-agent harness also controls which metadata enters the agent's context. We vary the harness to examine invocation under these different context-assembly mechanisms.

2.2 Repeated agent behavior

Repeated tool-use decisions provide a comparison for skill invocation. Yagubyan [2026] studied 6 models on 19 tasks with deterministic simulated tools. Tool-sequence consistency was associated with task correctness, while argument consistency was not significantly associated with correctness. Skill invocation also depends on a selection decision, but then exposes a procedure that may affect later decisions. The tool-use findings do not establish how repeatable skill invocation is or whether agents correct an initial routing error after reading a skill body. Yang et al. [2026] varied task wording on a fixed task set and measured changes in function-level code generation. Their design measures repeated code-generation outcomes; ours measures whether a skill is invoked.

Repeated evaluations also distinguish occasional success from reliable task completion. In CAR-bench's hardest task category, frontier agents succeeded in all 3 repeated trials less than 50% of the time, despite higher rates of success in at least 1 trial [Kirmayr et al. 2026]. Raj et al. [2026] proposed statistical methods for evaluating agent consistency and showed that trajectory measures detected failures that pass@1 obscured. We apply repeated trials to skill invocation.

3 Methodology

We vary the information available to an agent before it decides whether to invoke a skill. Each trial presents a task and a skill pool in a fresh copy of a synthetic repository. The design crosses 4 harnesses with 2 models and tests 9 hypotheses about metadata, instructions, and task prompts. H1–H8 measure invocation; H9 measures whether the agent performs an unrequested action. Table 3 summarizes the conditions and trial counts.

3.1 Definitions

SWEBOK: the Software Engineering Body of Knowledge, an IEEE Computer Society reference dividing the discipline into 18 knowledge areas [Washizaki 2024]. This study draws its 4 task categories from 4 of those areas: Construction, Testing, Quality, and Security. Software Engineering Operations and Software Configuration Management, despite ranking among the areas with the largest empirical skill share after Construction [Gao et al. 2026b], are excluded: Both

require artifacts (a deployment target, a monitoring system, a build pipeline) that this study's synthetic repository does not have.

Target skill: the skill manipulated by a hypothesis and evaluated for invocation. We author one per SWEBOK category, matched to that category's task.

Skill pool: the skills we place in the repository for a trial. By default, the pool contains 1 target skill and 3 fillers.

Filler: a skill in the trial's pool other than its target skill.

3.2 Repository, tasks, and skills

We author the tasks against the JavaScript HTTP server in the staged-project protocol of Lulla et al. [2026]. The server gains features in stages, and each stage starts from a partial implementation scaffolded to the point where that stage's feature must be added, with no residual package installations. We select the second stage because the initial server may be too simple for meaningful invocation tests, while later stages would make individual trials harder to interpret. We use the same snapshot throughout and do not repeat the trials in Python because programming language is not an experimental factor.

3.2.1 Authoring a task per category. Each category's task asks the agent to perform an action from that category on the same repository snapshot. We reset the repository before each trial so that tasks do not inherit changes from earlier trials.

Each task must map to a single category without an equally plausible interpretation under another. Tasks must also have comparable scope to reduce differences in routing difficulty caused by task complexity. We manually assess the task prompts and compare mean output-token counts across categories within each configuration, without setting a threshold for the difference. Each task identifies an action category and the specific action required. A language model drafts the tasks against these criteria, and the authors review and revise them.

3.2.2 Authoring skills. A language model drafts skills using examples sampled from skills.sh [Vercel 2026] to guide their wording. The authors check each draft's form, information content, and length. We estimate length with an open-source approximation of the harness tokenizers and accept drafts only within the interquartile range of token counts in a representative sample of skills.sh listings. We release the accepted names, descriptions, and task texts with the analysis code.

Each category has 1 target skill with a name, a description, and a body containing generic instructions. The body does not specify a task procedure because the primary outcome is invocation. In the default pool, the other 3 categories' target skills serve as fillers. The Baseline checks whether the agent reliably invokes the matching target in this pool before the hypothesis trials begin.

Unless a hypothesis specifies otherwise, the target is the only relevant skill in the pool, its name and description clearly match the task, and the project instruction file is empty. No additional instruction asks the agent to use skills.

3.3 Handling filler invocations

When a filler is invoked, we inspect the trial's response text and any reasoning summaries, check for accidental keyword overlap between the filler metadata and task, and assess whether the target remains relevant under the manipulation. We report this assessment with the filler invocation rate and retain the trial in every assigned comparison.

We treat accidental metadata overlap or a loss of target relevance as a possible problem with the experimental setup. Otherwise, the filler invocation is behavior observed under the condition. We retain both kinds of trial because excluding filler invocations could bias the comparison when a manipulation also reduces target invocation.

This treatment of filler invocations is the default for every hypothesis. Each subsection identifies any exception for a pool without fillers or an outcome that includes filler invocation.

We report invocations of skills that a harness ships with for each condition, separately from target and filler invocations.

3.4 Models and harnesses

Every trial runs non-interactively through a harness's software development kit (SDK) or command-line interface (CLI). We use 4 harnesses: Anthropic's Claude Code, OpenAI's Codex, Anomaly's OpenCode, and Mario Zechner's Pi. The models are Sonnet 5.5 and GPT-6.1 Sol. In every harness, each model runs at the default effort of its vendor's harness on October 10, 2026: medium for Sonnet 5.5 in Claude Code 2.1.296 [Anthropic 2026b] and low for GPT-6.1 Sol in Codex 0.162.1 [OpenAI 2026b]. We configure the effort level through each harness's native option and record the setting in effect for every trial using the harness's own terminology. Vendors' effort labels need not represent equivalent settings.

A harness assembles the agent's context and supplies its system instructions and tools. Crossing the models and harnesses lets us compare each model in its vendor's harness, in the other vendor's harness, and in model-agnostic harnesses. These comparisons include the combined effects of each harness's features.

Claude Code and Codex are vendor-specific harnesses. In an early-2026 survey of 906 readers of The Pragmatic Engineer, Orosz and Nilsson [2026] found that Claude Code was the most-used tool, having overtaken GitHub Copilot within 8 months of release. Codex, absent from the previous survey, had reached 60% of Cursor's usage. Each harness uses its vendor's model by default. Anthropic describes Claude Code's design in terms of giving agents the tools that programmers use [Anthropic 2025]. Codex emphasizes sandboxed execution and approval requirements [Bolin 2026]. The cross-vendor configurations test invocation when each harness runs the other vendor's model.

OpenCode and Pi are open-source, model-agnostic harnesses with different built-in capabilities. OpenCode documents subagents, skills, and Model Context Protocol (MCP) server support [OpenCode 2026d]. Pi supports on-demand skills and built-in MCP with Codemode, while leaving subagents, plan mode, permission prompts, task lists, and background shell execution to extensions or external workflows [Pi 2026a]. Their inclusion lets us compare invocation under these different sets of capabilities.

All 4 harnesses support the Agent Skills format [Anthropic 2026e; OpenAI 2026i; OpenCode 2026d; Pi 2026a]. Each loads skill names and descriptions into the agent's context and loads a skill's full `SKILL.md` only when the agent invokes it. We place the same `SKILL.md` contents in `.claude/skills/` for Claude Code and in `.agents/skills/` for Codex, OpenCode, and Pi, the project locations each harness documents. Each harness runs as its current release at the time of collection with its shipped defaults, including any skills it ships with, except for the specified manipulations. We record the skills each harness ships with at the start of collection and after each harness update.

Table 1 shows the 8 model and harness configurations. Six connect directly to the model provider. Claude Code with the OpenAI model and Codex with the Anthropic model use OpenRouter to translate between the vendors' protocols. These harnesses do not natively support the other vendor's application programming interface (API). Claude Code connects to OpenRouter's endpoint compatible with the Anthropic Messages API, and Codex connects to its endpoint compatible with the Responses API.

	Harness			
Model	Claude Code	Codex	OpenCode	Pi
Anthropic model	●	○	●	●
OpenAI model	○	●	●	●

Table 1. *The 8 harness-by-model configurations. Filled: direct to the model provider. Open: routed through OpenRouter.*

3.5 Baseline

The Baseline measures invocation for a clearly relevant target in each of the Construction, Testing, Quality, and Security categories [Washizaki 2024], with the other 3 category skills as fillers. The four-skill pool matches the median for repositories with 1,000–9,000 stars and more than 1 skill ($n = 313$) in Gao et al. [2026b]'s replication package [Gao et al. 2026c]. Target names identify the qualified subject and operation, while descriptions state the subject, operation, and when to use the skill. Table 2 gives the Construction example. The Baseline adds no manipulation.

Task	"The server currently discards everything after the request line, so it has no way to read a request's headers. Make it read the headers sent with each request."
Target skill name	Request header reading
Target skill description	Handles HTTP request headers. Reads and stores each header sent with the request. Use when a server needs to access header values.

Table 2. *Example task and target skill, for the Construction category.*

The Baseline checks task and skill pairings before the hypothesis trials. Only Claude Code with the Anthropic model must pass the admission gate. We use this as the reference configuration because Anthropic introduced the skills mechanism and Claude Code provides its native implementation. We require a near-ceiling invocation rate in this configuration to reduce ambiguity between poor task-skill matching and an effect of the manipulation. Each target must be invoked reliably with the other 3 categories' skills present as fillers.

The gate requires 0 misses across 60 trials. For a fixed sample, this gives an exact one-sided 95% Clopper–Pearson lower bound above the target invocation probability of 0.95 [Clopper and Pearson 1934]. The required sample size is $n = \ln(1 - 0.95)/\ln(0.95) \approx 58.4$; we use 60. Yeon et al. [2025] used a similar bound for tool-selection satisfaction probability. If a category fails, we check for wording or setup problems before revising or replacing its task or skill. A revised task repeats the gate in its own category. A revised skill repeats the gate in all 4 categories, because each target skill is a filler in the other 3 pools. We retain and report the trials of every gate, failed or passing. Tasks and skills are fixed once all 4 categories pass. We then run 60 new Baseline trials per category in this configuration before any hypothesis trials begin, and later comparisons use these trials as the Baseline. Conditioning admission on 0 misses biases the observed Baseline invocation rate upward, which would raise each support criterion's chance probability above the values in Table 10. We run the Baseline in the other 7 configurations without an admission requirement.

3.6 Hypotheses

Every hypothesis below is non-directional: No test predicts which condition produces a higher or lower rate, only whether the manipulated factor has an effect.

Each support criterion compares a hypothesis's conditions with a reference condition. Where a hypothesis crosses 2 factors, we compare the levels of each factor with the other factor held fixed and report these differences and the interaction descriptively, with 95% confidence intervals.

	Group	Tests	Factor(s) and levels	Trials
Baseline		Each category's target skill is invoked reliably with no manipulation applied.	None	2,160
H1	Skill/pool	Metadata specificity and field affect invocation rate.	Name level × description level (4 × 4)	5,120
H2	Skill/pool	Skills with overlapping scopes in a pool affect invocation rate.	Competitor name/description pairing (3)	960
H3	Skill/pool	Conflicting task categories in a skill's name and description affect invocation rate.	Conflict pairing × task category (6 × 2)	960
H4	Skill/pool	The number of filler skills in the pool affects invocation rate.	Skill pool size (4: 1, 6, 14, 140)	1,280
H5	Environment	The type of conflicting instruction and its delivery channel affect invocation rate.	Signal type × delivery channel (3 × 5)	4,800
H6	Environment	Instruction pressure to use skills affects invocation rate.	Pressure strength × target availability (4 × 2)	2,560
H7	Environment	How a task is framed affects invocation rate.	Task framing (5)	1,920
H8	What's asked	Combining tasks in a single prompt affects invocation of each task's skill.	Task composition (2: solo, composed)	320
H9	What's asked	A skill's presence affects whether its action occurs on tasks that do not explicitly request or require it.	Skill presence × action (2 × 2)	320
Total				**20,400**

Table 3. *All planned trials in the study, by the Baseline and each of the 9 hypotheses, with their conditions.*

For most hypotheses, the trial count is the number of conditions multiplied by 4 task categories, 8 configurations, and k trials. H3's 12 conditions already include the category comparisons. H7 adds a reference condition to its 5 variants. H8 uses 4 task orderings and collects only composed trials, reusing the Baseline for solo trials. H9 has 4 conditions in Construction only. The Baseline uses 60 trials per category and configuration, and 60 admission trials per category in Claude Code with Sonnet 5.5.

3.6.1 H1 (Skill Metadata). Metadata specificity and the field carrying that information affect skill invocation rate.

A skill author must decide what to put in the name and description, because the agent selects a skill before reading its body. Editing the body alone cannot change that initial decision when the metadata and other inputs remain unchanged. H1 varies the detail in each field to test its effect on invocation. We also report descriptively whether a detailed description compensates for a less informative name, or the reverse.

Each field has 4 levels, with each successive level adding information. We cross the name and description levels to form the 16 conditions in Table 4.

The lowest description level (1) uses "None" because the platform rejects an empty field [Anthropic 2026e]. The lowest name level (A) uses a shared opaque token.

Description level (2) names the subject: "Handles HTTP request headers." Level (3) appends the operation: "Reads and stores each header sent with the request." Level (4) appends when to use the skill: "Use when a server needs to access header values." Levels (3) and (4) retain the sentences from the preceding level. Level (4) follows Anthropic's guidance to state both what a skill does and when to use it [Anthropic 2026e].

Name level (B) identifies the subject, "Headers"; level (C) adds a qualifier, "Request headers"; and level (D) adds the operation, "Request header reading".

Pairing name level (D) with description level (4) reproduces the Baseline's target metadata. Lower levels remove information from one or both fields. Table 4 identifies each condition by its name and description levels.

	Description			
Name	(1) None	(2) Subject	(3) + Operation	(4) + When-to-use
(A) Opaque	A1	A2	A3	A4
(B) Subject	B1	B2	B3	B4
(C) + Qualifier	C1	C2	C3	C4
(D) + Operation	D1	D2	D3	**D4**

Table 4. *H1 (Skill Metadata): name level (row, A–D) and description level (column, 1–4) identifying each pairing.*

H1's support criterion is met for a category and configuration if the invocation rates of at least 3 of the 15 cells other than D4 differ significantly from the Baseline. We compare name levels within each column of Table 4, where the description level is fixed, and description levels within each row, where the name level is fixed.

Factors	Name level × description level
Levels	4 × 4, defined in Table 4
Controls	Skill names do not include names of implementation techniques or mechanisms
Support criterion	Invocation rates of at least 3 of the 15 cells other than D4 differ significantly from the Baseline (Boschloo's exact test)

3.6.2 H2 (Skill Competition). Skills with overlapping scopes in a pool affect invocation rate.

H2 tests how competition from near-duplicate skills changes invocation. Each pool contains 4 entries describing the same underlying skill, with no unrelated fillers. Every entry corresponds to an H1 condition in which it was the only relevant skill among 3 fillers. Comparing its H1 and H2 invocation rates measures the effect of replacing those fillers with competitors.

All 3 pool configurations (P1–P3) use name levels (A)–(D), giving each entry in a pool a distinct name. They differ in how the description levels are paired with those names.

P1 uses the same level-(4) description for all 4 entries and varies the name from (A) to (D). Differences in invocation rates measure the contribution of the name when the description is fixed at the documented standard.

P2 increases name and description detail together, using A1, B2, C3, and D4. Each entry adds information to both fields.

P3 pairs more detailed names with less detailed descriptions, using A4, B3, C2, and D1. Table 5 shows the entries in all 3 pool configurations.

	Description			
Name	(1) None	(2) Subject	(3) + Operation	(4) + When-to-use
(A) Opaque	P2	—	—	P1, P3
(B) Subject	—	P2	P3	P1
(C) + Qualifier	—	P3	P2	P1
(D) + Operation	P3	—	—	P1, P2

Table 5. *H2 (Skill Competition): which pool configuration draws an entry from each (name, description) pair in H1's grid (Table 4, same rows and columns). A pair listing 2 pool configurations is reused in both.*

More than 1 entry can be invoked in an H2 trial. The inclusive invocation measure counts each entry that is invoked, regardless of other invocations or their order. We also record sole invocation, which counts an entry only when no other entry is invoked in the trial. We retain both measures because some workflows allow several skills to be used together, while others require exactly one.

For each skill entry, we compare its 10 H1 reference trials with its 10 H2 competition trials using Boschloo's exact test. There are 12 comparisons, some sharing H2 trials or H1 reference data. H2's support criterion is met for a category and configuration if the inclusive invocation rates of at least 3 of the 12 entries differ significantly between the 2 conditions.

Factors	Name/description pairing across near-duplicate competitors
Levels	3 pool configurations (P1–P3), defined in Table 5
Controls	4 near-duplicate entries per pool; no fillers
Support criterion	Inclusive invocation rates of at least 3 of the 12 entries differ significantly between the H1 reference and H2 competition conditions (Boschloo's exact test)

3.6.3 H3 (Metadata Conflict). Conflicting task categories in a skill's name and description affect invocation rate.

Independent revisions can leave a skill's name and description pointing to different tasks. H3 tests whether conflicting categories in these fields affect invocation.

We create conflicting metadata by taking a name from one task category and a description from another. Both fields retain the Baseline's level of detail: The name gives a qualified subject and operation, and the description gives a subject, operation, and when to use the skill. Only the category differs.

We test all 6 pairs of SWEBOK categories (Table 6). For each pair, one entry combines the first category's name with the second category's description; the other combines the second category's name with the first category's description. The pool also contains 2 fillers from the categories outside the pair, with names and descriptions that agree.

We run each category's task with this pool. For one task, the first entry matches by name and the second by description; for the other task, these roles reverse.

	Testing	Quality	Security
Construction	1	2	3
Testing		4	5
Quality			6

Table 6. *H3 (Metadata Conflict): the 6 conflict pairings. Each conflict pairing is run twice, once under each category's own task, using the same 2 entries both times.*

Both conflicting entries can be invoked in the same trial. We use H2's inclusive and sole-invocation measures and report whether the name-matching entry, the description-matching entry, both, or neither was invoked. We compare the 2 entries' rates descriptively to show which field better predicts selection when the fields disagree.

H3's support criterion is met for a category and configuration if the invocation rates of at least 2 of the 6 entries run under the category's task differ significantly from that category's Baseline.

Factors	Conflict pairing; task
Levels	6 conflict pairings, defined in Table 6, each run twice
Controls	Fillers fixed per conflict pairing and non-conflicting
Support criterion	Invocation rates of at least 2 of the 6 entries run under a category's task differ significantly from that category's Baseline (Boschloo's exact test)

3.6.4 *H4 (Skill Pool). The number of filler skills in the pool affects skill invocation rate.*

Adding skills to a repository may change how often existing skills are invoked. H4 tests this effect at 4 pool sizes.

Pool size includes the target and all fillers. We choose 3 levels from the skills-per-repository distribution in Gao et al. [2026b]'s replication package [Gao et al. 2026c] and add a larger stress condition. The released dataset excludes repositories above the authors' outlier cutoff of 18 skills. Fillers added to the Baseline pool take their names and descriptions from skills.sh [Vercel 2026] listings whose scope does not overlap any of the 4 target skills. Their bodies contain generic instructions in the format of the target skills' bodies. These fillers differ in content from the Baseline's fillers, which are the other categories' target skills, but none competes with the target task. We select only listings within the token-length limits of the authored skills.

Level 1 contains only the target, corresponding to the distribution's 25th percentile. Level 2 adds 2 fillers to the Baseline pool, for 6 skills at the 75th percentile. Level 3 adds 8 fillers to Level 2, for 14 skills, just above the filtered distribution's outlier threshold of 13.5. Level 4 adds 126 fillers to Level 3, for 140 skills, as an intentionally extreme stress condition, testing whether effects observed at smaller sizes continue or saturate.

H4's support criterion is met for a category and configuration if the invocation rates of at least 2 of the 4 levels differ significantly from the Baseline.

Factors	Pool size
Levels	(1, 6, 14, 140), defined above
Controls	Fillers (non-competing); pool delivery location
Support criterion	Invocation rates of at least 2 of the 4 levels differ significantly from the Baseline (Boschloo's exact test)

3.6.5 H5 (Signal and Source). The type of conflicting instruction and its delivery channel affect skill invocation rate.

Project instructions can discourage skill use directly or contradict the requested task. We call the added instruction a signal and vary both its type and its delivery channel to test their effects on invocation in H5.

We use 3 signal types. Type (1) discourages skills and tools with the same instruction in every category: "Do not use skills or tools for this if you can do it yourself." Type (2) states a category-specific project convention that contradicts the requested task, without mentioning skills or tools. The Construction example is "Request handling reads only the request line and does not read or store headers." Type (3) concatenates the type-(2) instruction and the type-(1) instruction, in that order, without changing either sentence.

The delivery channels are (A) a code comment in a file that the agent works on, (B) a project document that it is directed to read, such as `CONTRIBUTING.md`, (C) the project instruction file loaded by the harness, such as `AGENTS.md` or `CLAUDE.md`, (D) the task prompt, and (E) the system prompt.

Table 7 crosses the signal types and channels. The Baseline provides the comparison without a restrictive signal. Channel (C) places the signal in the otherwise empty project instruction file.

	Channel				
Signal type	(A) Comment	(B) Doc file	(C) Instruction file	(D) Prompt	(E) System prompt
(1) Discouraging	A1	B1	C1	D1	E1
(2) Conflicting	A2	B2	C2	D2	E2
(3) Both	A3	B3	C3	D3	E3

Table 7. *H5 (Signal and Source): the crossed conditions, e.g. E3 = both lines via the system prompt.*

Only channel (E) changes the system prompt. Claude Code and Pi add text to their system prompt with the `--append-system-prompt` flag [Anthropic 2026a; Pi 2026c], and OpenCode adds it through the `system` field of a session request [OpenCode 2026a]. Codex does not natively support appending to its system prompt and only allows its built-in instructions to be replaced [OpenAI 2026f]. For Codex, we copy the built-in instructions from its first model request for each model and version, append the signal, and supply the result as the replacement. Before collection, we compare each harness's first model request with and without the signal in every configuration and confirm that the signal is the only difference.

H5's support criterion is met for a category and configuration if the invocation rates of at least 3 of the 15 conditions differ significantly from the Baseline. We compare signal types within each column of Table 7, where the channel is fixed, and channels within each row, where the signal type is fixed. In channels (A) and (B), the agent sees the signal only when it reads the file containing it, so we record whether that read precedes its first skill invocation.

Factors	Signal type × delivery channel
Levels	3 × 5
Controls	Signal wording, fixed per type across channels; filler skills
Support criterion	Invocation rates of at least 3 of the 15 conditions differ significantly from the Baseline (Boschloo's exact test)

3.6.6 H6 (Instruction Pressure). Instruction pressure to use skills affects skill invocation rate.

Instructions to use skills are a possible remedy for missed invocations, but may also prompt an agent to invoke an irrelevant skill. H6 tests how instruction strength changes target and filler invocation rates.

For each category and configuration, H6 uses the H1 entry with the observed invocation rate closest to 50%. If several entries are equally close, we select the first in alphabetical order of its condition label in Table 4. This selection leaves room to observe either an increase or a decrease. We test it in a pool with 3 fillers and compare instruction levels by the target invocation rate. We also test the same instructions with only the 3 fillers present, measuring the proportion of trials in which any filler is invoked. The selected H1 entry may differ between configurations.

We apply 4 instruction levels in both target-availability conditions. Level 1 adds no instruction. Level 2 adds "Use skills where relevant." Level 3 adds "You must use available skills." Level 4 adds "You must use the `<skill-name>` skill." Levels 2 and 3 use generic instructions; level 4 tests explicit naming, even when the named target is absent.

H6's support criterion is met for a category and configuration if at least 2 of its 6 comparisons with level 1 are significant: the target invocation rates at levels 2–4 with the target present, and the filler invocation rates at levels 2–4 with the target absent.

We analyze the target-present and target-absent conditions separately. Within each configuration, we also compare reasoning-token counts between the 2 conditions as a measure of deliberation cost, grouping trials by whether any skill is invoked.

The target-absent condition is an exception to the default filler handling. Because no pool member fits the task, we measure filler invocation without applying the filler check. Trials in which more than 1 filler is invoked are reported as such.

Factors	Instruction-pressure strength × target availability
Levels	Instruction levels 1–4 × (present, absent)
Controls	3 fillers per pool
Support criterion	At least 2 of 6 comparisons with level 1 are significant: target invocation rates at levels 2–4 (target present) and filler invocation rates at levels 2–4 (target absent) (Boschloo's exact test)

3.6.7 H7 (Task Framing). How a task is framed affects skill invocation rate.

A skill author cannot control how a user phrases the task. H7 tests whether changes in task wording affect invocation when the skill metadata is fixed.

H7 uses 5 variants of the Baseline task prompt and the same H1 selection rule as H6: the entry with an observed invocation rate closest to 50% in each category and configuration. A reference rate near 50% leaves room to detect increases that a near-ceiling Baseline could obscure. As the reference, we collect k new trials of the selected entry with the unmodified Baseline prompt.

Each variant makes a single edit to the Baseline prompt and is compared with these reference trials. Condition (1) removes the action clause, leaving only the symptom, to test an underspecified request. Condition (2) names a technique to test whether specifying the implementation changes the invocation rate. Condition (3) adds urgency. Condition (4) asks the agent to gather information before acting. Condition (5) adds an unrelated action to test invocation when the prompt contains another request. Table 8 gives the Construction examples.

#	Example (Construction)
1	Baseline minus the action clause: "The server currently discards everything after the request line, so it has no way to read a request's headers."
2	Baseline plus: "...by parsing each line into a name and value, split on the first colon."
3	Baseline plus: "Please fix this as soon as possible."
4	Baseline plus: "Collect all the relevant information about this before acting."
5	Baseline plus: "Also add a `.gitignore` entry for common editor swap files."

Table 8. *H7 (Task Framing): example task text for all 5 conditions, shown for the Construction category.*

H7's support criterion is met for a category and configuration if the invocation rates of at least 2 of the 5 variants differ significantly from the reference.

Factors	Task wording/framing
Levels	5 conditions, defined above
Controls	Skill name and description, identical across all conditions
Support criterion	Invocation rates of at least 2 of the 5 variants differ significantly from the reference (Boschloo's exact test)

3.6.8 H8 (Task Composition). Combining tasks in a single prompt affects invocation of each task's skill.

H8 tests whether an agent still invokes each relevant skill when a user combines several tasks in a single prompt. We concatenate the 4 category tasks and compare new trials for this composed condition with the Baseline's solo trials. The skill pool remains unchanged, but every entry now matches a requested task. We therefore count invocations toward each skill's rate and the joint pass rate without applying the filler check.

Task order within the composed prompt is balanced by a Latin square: Each task occupies each position exactly once across the orderings in Table 9.

Ordering	Position 1	Position 2	Position 3	Position 4
1	Construction	Testing	Quality	Security
2	Testing	Quality	Security	Construction
3	Quality	Security	Construction	Testing
4	Security	Construction	Testing	Quality

Table 9. *H8 (Task Composition): the 4 Latin-square task orderings.*

We estimate each skill's invocation rate from all 40 composed trials per configuration: 10 repetitions of each ordering in Table 9. Boschloo's exact test compares each rate with the same skill's Baseline solo rate. H8's support criterion is

met for a configuration if the rates of at least 2 of the 4 skills differ significantly from their Baseline solo rates. We also report the proportion of composed trials in which all 4 skills are invoked as a descriptive joint pass rate. For each composed trial, we describe the order in which skills are invoked, including which skill is invoked first.

Factors	Task composition
Levels	Solo (the Baseline's trials), composed (the Baseline's tasks concatenated, order balanced per Table 9)
Outcome	Each skill's own invocation rate under composition
Controls	Skill pool and task text (reused from the Baseline); skill name and description
Support criterion	Composition-condition rates of at least 2 of the 4 skills differ significantly from their Baseline solo rates (Boschloo's exact test)

3.6.9 H9 (Task Scope). A skill's presence affects whether its action occurs on tasks that do not explicitly request or require it.

Skill availability may affect agent behavior even when the task does not call for the skill: its metadata is visible before invocation, and its procedure may influence subsequent actions if invoked. H9 therefore tests whether making a skill available changes the occurrence of an action that the task does not request. We examine running the project's test suite and updating its docstrings after the unchanged Construction task (Table 2). Both actions are plausible after a code change, and testing both reduces reliance on a single action that may already occur in nearly every trial without a skill. For each action, we compare a pool containing only the relevant skill with an empty pool. For H9, we add a test script with tests for the existing routes and a docstring on one existing function in a file that the Construction task modifies. Both arms use this version of the repository. We use Construction because the other category tasks do not require implementing new functionality. Test execution is coded from the tool-call trace as a command that runs the test script. A docstring update is coded from the tool-call trace as an edit that adds or changes a docstring. Both actions are coded whether or not the skill was invoked.

The comparison estimates the effect of making a skill available. Its name and description are visible before invocation, and loading its body may later affect the agent's actions. Counting actions regardless of invocation includes both possibilities.

H9's support criterion is met if action occurrence differs significantly between skill-present and skill-absent trials, with the two actions pooled into a single comparison of 20 trials per condition. This pooled comparison gives equal weight to the two actions; we also report the effect for each action descriptively. Each trial is scored only for the action it was run for. We also report invocation descriptively.

Factors	Skill presence × action
Levels	(Absent, present) × (test execution, docstring update)
Controls	No fillers
Support criterion	Occurrence rate differs significantly between skill-absent and skill-present, both actions pooled (Boschloo's exact test)

3.7 Analysis plan

3.7.1 Statistical tests. Every confirmatory outcome in the study is binary: whether any skill was invoked, whether the invoked skill was the target, whether more than 1 distinct skill was invoked, and whether an action occurred. Every

comparison specified by a hypothesis's support criterion is tested separately within each of the 8 configurations of Table 1, using that hypothesis's category grouping. H8 evaluates its 4 skills together within each configuration, and H9 uses Construction only.

Every comparison is between independent groups and uses Boschloo's exact test at $\alpha = 0.05$. If both groups contain only successes or only failures, $p = 1$. Boschloo's two-sided p-value doubles the smaller one-sided p-value. The 95% confidence interval combines two one-sided 97.5% bounds from the unconditional exact test ordered by Fisher's mid-p-value [Fay and Hunsberger 2021].

The design favors breadth: it tests 9 hypotheses across 4 task categories and 8 configurations, with k trials per condition, for 20,400 trials including the Baseline. Every comparison named in a support criterion has equal standing, and the study sets no minimum difference worth detecting. With $k = 10$, two conditions differ significantly only when their rates differ by at least 0.4, and a condition compared with a Baseline of 60 invocations in 60 trials differs significantly at a rate of 0.8 or lower. We compute the chance probabilities in Table 10 exactly from the comparisons each support criterion uses, evaluating comparisons that share reference trials jointly, and take the largest value over the common invocation rate.

To complement the confirmatory rate comparisons with qualitative evidence about agent behavior, for each hypothesis we select for exploratory review up to 2 comparisons named in its support criterion, taking those with the smallest p-values between 0.05 and 0.20 across all categories and configurations. Where comparisons tie, we select by which trials' traces show the most informative handling of the target skill. We review the tool-call traces, response text, and any reasoning summaries of their trials for how the agent handled the target skill. These reviews are reported separately from the support counts as exploratory results.

Hypothesis	Support criterion	Reference rate	Power of 1 comparison at 0.25	Trials for 80% power	Met by chance	Expected count
H1	At least 3 of 15 (10 vs. 60)	0.95	53.2%	24	4.7%	1.5
H2	At least 3 of 12 (10 vs. 10)	0.50	18.2%	60	1.5%	0.5
H3	At least 2 of 6 (10 vs. 60)	0.95	53.2%	24	3.5%	1.1
H4	At least 2 of 4 (10 vs. 60)	0.95	53.2%	24	1.6%	0.5
H5	At least 3 of 15 (10 vs. 60)	0.95	53.2%	24	4.7%	1.5
H6	At least 2 of 6 (10 vs. 10)	0.50	18.2%	60	4.2%	1.3
H7	At least 2 of 5 (10 vs. 10)	0.50	18.2%	60	3.9%	1.3
H8	At least 2 of 4 (40 vs. 60)	0.95	91.4%	24	1.3%	0.1
H9	1 pooled (20 vs. 20)	0.50	34.1%	60	4.6%	0.4

Table 10. *Power and chance support per hypothesis (Boschloo's exact test, $\alpha = 0.05$).*

Each support criterion requires the smallest number of significant comparisons that keeps its chance probability within a category and configuration, or within a configuration for H8 and H9, at or below 5% (Table 10). We therefore do not adjust individual p-values for the number of comparisons. Each hypothesis reports the number of categories and configurations in which its support criterion is met, out of 32 (8 for H8 and H9), alongside the number expected by chance.

3.7.2 Measures. We collect the measures in Table 11 for every trial. They support further analysis of agent behavior but have no separate support criteria.

Measure	Definition
First invocation turn	Turn index of the first skill invocation.
Token consumption	Total tokens consumed in a trial, measured and compared within a task category.
Reasoning tokens	Reasoning tokens the model generated in a trial, as reported by the harness, measured and compared within a task category and configuration.
Turns to completion	Total agent turns in a trial, measured and compared within a task category.
Output-produced flag	Whether the trial produced any task output.
File-read flag	Whether the tool-call trace shows the agent reading a project file before its first skill invocation; for trials without a skill invocation, before its first edit to a project file.

Table 11. *Measures collected on every trial.*

3.7.3 Reporting commitments. For every comparison specified by a support criterion, we report the difference in rates with its p-value and a 95% confidence interval. Analyses beyond the measures specified in this study will be labeled exploratory. For each conclusion drawn from response text or reasoning summaries, a second author reviews the same trials without seeing the first author's conclusion. Both authors discuss any disagreement and record the final decision. We will report and explain deviations from this registered report.

3.8 Data collection

Each trial runs unattended through the harness CLI or SDK in a fresh Docker container, with a single task prompt and no working tree, cache, or conversation state retained from earlier trials. The working tree contains only a copy of the selected project directory. Before each trial, we restore the category's untouched snapshot, write the skill pool, and place the artifacts required by the condition.

We collect k trials per experimental condition and 60 per Baseline category in each of the 8 configurations (Table 1). We retain each session that terminates abnormally as a recorded attempt excluded from the analysis, and rerun the trial [Lulla et al. 2026]. We classify each such attempt as an infrastructure failure if it fails before its first model request or ends on a failed model request, and as an agent failure otherwise, and report the number of each per condition. A trial that terminates abnormally on 3 consecutive attempts is reported as missing.

We use the same language runtime throughout collection. Harness settings other than effort remain at their defaults except for specified manipulations. We pin the harness version for the duration of each configuration; if the harness version changes, we repeat that configuration. Within each configuration, we collect the Baseline first, followed by H1 through H9, completing each category before proceeding to the next. Within a hypothesis and category, the trials of all conditions run in an order shuffled with a recorded random seed. All trials for one configuration finish before another begins. Claude Code with the Anthropic model is collected first, because its admission gate fixes the tasks and skills. This order lets later hypotheses use earlier results and allows us to correct detected setup defects before proceeding. If we correct a setup defect, we repeat every trial collected under that setup in the affected configuration.

Each trial records the repository snapshot hash, task prompt hash, environment-reset confirmation, model version string, harness version string, and, for each model request, the endpoint called and the provider that served it. We report the model and harness versions with each configuration's results. Anthropic states that the weights and configuration

associated with a model identifier remain fixed [Anthropic 2026c]. The OpenAI model has no published dated snapshot; its identifier may refer to different versions during collection [OpenAI 2026g]. A detected model or harness version change requires repeating that configuration.

We capture tool-call traces as the evidence for confirmatory outcomes, including skill invocations, their turns, and file access. We also retain the agent's response text, including text written between tool calls, to distinguish clarification requests from trials in which the agent does nothing. Anthropic and OpenAI models return only a summary of their reasoning, and only when the request asks for one [Anthropic 2026f; OpenAI 2026h]. With default settings, Codex, OpenCode, and Pi request reasoning summaries from the OpenAI model, and Pi also requests them from the Anthropic model [OpenAI 2026c; OpenCode 2026b; Pi 2026b,d]. Claude Code in non-interactive mode and OpenCode request none from the Anthropic model, which returns no reasoning text by default [Anthropic 2026f; OpenCode 2026b]. We record for each trial, including those routed through OpenRouter, whether reasoning summaries were returned. We review the response text and any reasoning summaries to investigate filler invocation and the comparisons selected for exploratory review. Neither determines confirmatory outcomes.

A skill is invoked when a tool call returns its `SKILL.md` content to the agent or to a subagent it starts. In Claude Code and OpenCode, this is a call to the harness's skill tool naming the skill; in Codex and Pi, it is a successful file read or shell command that names the skill's `SKILL.md` path and returns at least part of its contents [Anthropic 2026e; OpenAI 2026d; OpenCode 2026d; Pi 2026a]. Either event counts in every harness, including partial reads, and a call that returns an error is not an invocation. We record every distinct skill invoked in a trial and whether the agent or a subagent invoked it. Repeated invocation of a skill counts as 1 distinct entry, and trials with multiple entries retain all of them. Invocation order is not an outcome, tiebreaker, or covariate because it does not establish preference. We log the turn of the first invocation and, for a subagent's invocation, take the agent turn that started the subagent. We report the proportion of trials with multiple distinct invocations and the proportion of invocations made by a subagent in each condition.

3.9 Outputs

On publication, we release the analysis code, per-trial records, authored skill names and descriptions, task texts, and each trial's response text and reasoning summaries. Trial records include outcome coding, repository and prompt hashes, environment-reset confirmation, and model and harness version strings. We deposit these artifacts in a public archive with a persistent identifier and permissive license, provide a copy to reviewers at submission, and cite the archive in the completed manuscript. The shared base repository will be released separately with its protocol.

4 Estimated cost

We estimate collection costs from a cost pilot [Lee 2026]. In the first measurement, we sent the prompt "Reply with OK." three times to each configuration in an empty repository and recorded the input tokens of the first API call, which carry the harness's system prompt and tool definitions (Table 12). Every API call in a session carries this overhead, so session costs include it.

Model	Claude Code	Codex	OpenCode	Pi
Sonnet 5.5	19,434	15,556	7,443	2,300
GPT-6.1 Sol	19,953	14,429	4,200	1,160

Table 12. *Median input tokens of the first API call, by configuration.*

Table 13 gives the estimate from 3 sessions the pilot ran in each of the 8 configurations. In each session, we asked the agent to add request parsing to a working build of the second-stage HTTP server and allowed the session to run until the harness exited. The pilot used the routes in Table 1, except that OpenCode and Pi with Sonnet 5.5 were routed through OpenRouter. We price every API call in each session, including cached input, at the model's standard API prices. Both models have the same standard prices per million tokens, $2 for input and $10 for output; we price cached input at $0.20 per million tokens for Sonnet 5.5 and $0.10 for GPT-6.1 Sol, and cache writes at $2.50 for both [Anthropic 2026d; OpenAI 2026a]. Each configuration's median and mean session costs give its cost per trial, and its study cost is that cost multiplied by its share of the 20,400 trials: 2,760 for Claude Code with Sonnet 5.5 and 2,520 for each other configuration. Totals are summed before rounding. The repository reports the input, cached-input, and output tokens of every API call, the first-call measurements, and the prices used.

		Session cost ($)		Study cost ($)	
Model	**Harness**	**Median**	**Mean**	**Median**	**Mean**
Sonnet 5.5	Claude Code	0.150	0.149	413	410
	Codex	0.112	0.114	282	288
	OpenCode	0.080	0.080	202	200
	Pi	0.054	0.092	136	231
GPT-6.1 Sol	Claude Code	0.153	0.155	384	390
	Codex	0.062	0.060	156	152
	OpenCode	0.064	0.061	161	155
	Pi	0.055	0.055	138	139
Total				**1,873**	**1,966**

Table 13. *Estimated collection cost by configuration, from the cost pilot's session costs.*

5 Threats to validity

Construct Validity. H1–H8 measure invocation outcomes across repeated trials; they do not assess whether an invoked procedure is followed correctly or improves the work. For H1–H8, a trial where the target skill is invoked and the task is completed well scores identically to one where it is invoked and the task is completed poorly. H9 instead measures the effect of skill availability on action occurrence, without isolating whether that effect depends on invocation.

Internal Validity. Adding metadata detail or pool entries increases both information content and text length, which the design does not separate. Padding shorter conditions to equalize length would itself add content that could affect routing.

The target's position in the skill list shown to the agent may also affect invocation. Liu et al. [2024] found lower retrieval accuracy for entries in the middle of serialized key-value lists than at either end. OpenCode sorts the list by skill name; the cited Codex and Pi source files sort directory entries without establishing the order of the full list [OpenAI 2026e; OpenCode 2026c; Pi 2026e], and we have not established Claude Code's ordering rule. The design does not control list position.

Each harness combines a system prompt, tool set, context assembly, and skill presentation. Differences between harnesses may reflect any of these features, which the study does not isolate. The cross-vendor configurations also use an aggregator to translate requests between vendor APIs [OpenRouter 2026]. Results for these configurations may therefore depend on translation as well as the model and harness.

We collect the 8 configurations sequentially. Anthropic guarantees fixed weights and configuration for a model identifier [Anthropic 2026c], but this does not establish that serving infrastructure, routing, or sampling stays unchanged. Without a published dated snapshot for the OpenAI model [OpenAI 2026g], comparisons between its harness configurations may also reflect model updates. These risks apply to comparisons across configurations, including counts of configurations supporting a hypothesis. In the cross-vendor configurations, differences between upstream providers selected by the aggregator may also affect results.

Only Claude Code with the Anthropic model must pass the admission gate. We revise tasks and skills until this configuration passes, so their wording may suit it better than the other 7 configurations and raise its invocation rates. This may favor that configuration in comparisons across configurations.

External Validity. All study objects are synthetic. The HTTP server has no third-party dependencies or project history, and its functionality is limited to serving requests. These omissions remove context that may affect routing in a deployed repository. We draft skills using marketplace examples but admit them only when their fields contain the information specified for their conditions. The measured rates therefore apply to this repository, the authored pool, and the single task in each category. They do not estimate invocation rates in established codebases or pools taken from a marketplace.

Models may encounter tool interfaces and skill formats during training. For example, R2E-Gym trained agents on trajectories using a file editor, a search tool, a bash terminal, and a submit action [Jain et al. 2025]. The study's skills are newly authored, but similar metadata, procedures, or formats may occur in the models' training data.

Conclusion Validity. Identical inputs do not guarantee identical outputs from stochastic coding agents. Ouyang et al. [2025] found that, for 47.6–75.8% of tasks, no test input produced equivalent outputs across all 5 programs generated from identical prompts. Bjarnason et al. [2026] found variation in multi-step agent runs, with single-run pass rates differing by 2.2–6.0 percentage points depending on which run was reported. Our trials similarly involve multiple agent steps, with sampling parameters such as temperature left at the harness or vendor default, and Blyth et al. [2024] showed that temperature alone measurably shifts pass rate and output diversity in code generation. Each additional repetition adds a trial, and its cost, to every condition in every category and configuration. Most conditions therefore have 10 repetitions, which limits the precision of rate estimates; the estimates may differ in an independent replication.

The power estimates in Table 10 are lowest for the 10-versus-10 comparisons used in H2, H6, and H7. A non-significant result does not establish the absence of an effect. In H9, opposite changes in test execution and docstring updates can offset each other in the pooled comparison, so we report the difference for each action descriptively. Comparisons that narrowly miss significance receive the exploratory review described in Section 3.7.

References

Anthropic. 2025. *Building Agents with the Claude Agent SDK.* Retrieved August 14, 2026 from https://claude.com/blog/building-agents-with-the-claude-agent-sdk

Anthropic. 2026a. *CLI Reference.* Retrieved August 25, 2026 from https://code.claude.com/docs/en/cli-reference

Anthropic. 2026b. *Model configuration.* Retrieved October 5, 2026 from https://code.claude.com/docs/en/model-config

Anthropic. 2026c. *Model IDs and versioning.* Retrieved September 1, 2026 from https://platform.claude.com/docs/en/about-claude/models/model-ids-and-versions

Anthropic. 2026d. *Pricing.* Retrieved October 5, 2026 from https://platform.claude.com/docs/en/about-claude/pricing

Anthropic. 2026e. *Skill Authoring Best Practices.* Retrieved July 30, 2026 from https://platform.claude.com/docs/en/agents-and-tools/agent-skills/best-practices

Anthropic. 2026f. *Thinking.* Retrieved October 4, 2026 from https://platform.claude.com/docs/en/build-with-claude/thinking

Bjarni Haukur Bjarnason, André Silva, and Martin Monperrus. 2026. On Randomness in Agentic Evals. arXiv:2602.07150 doi:10.48550/arXiv.2602.07150 Published, ICLR 2026 Workshop on Agents in the Wild.

Scott Blyth, Markus Wagner, and Christoph Treude. 2024. Creative and Correct: Requesting Diverse Code Solutions from AI Foundation Models. arXiv:2403.13259 doi:10.48550/arXiv.2403.13259 Published, FORGE 2024 (AI Foundation Models and Software Engineering Workshop, ICSE).

Michael Bolin. 2026. *Unrolling the Codex Agent Loop.* Retrieved August 14, 2026 from https://openai.com/index/unrolling-the-codex-agent-loop/

C. J. Clopper and E. S. Pearson. 1934. The Use of Confidence or Fiducial Limits Illustrated in the Case of the Binomial. *Biometrika* 26, 4 (1934), 404–413. doi:10.1093/biomet/26.4.404

Michael P. Fay and Sally A. Hunsberger. 2021. Practical Valid Inferences for the Two-Sample Binomial Problem. *Statistics Surveys* 15 (2021), 72–110. doi:10.1214/21-SS131

Matthias Galster, Seyedmoein Mohsenimofidi, Levi Böhme, Jai Lal Lulla, Muhammad Auwal Abubakar, Christoph Treude, and Sebastian Baltes. 2026a. A Dataset of Agentic AI Coding Tool Configurations. In *Proceedings of the 3rd ACM International Conference on AI-Powered Software (AIware '26).* Association for Computing Machinery, New York, NY, USA, 314–322. arXiv:2605.08435 doi:10.1145/3805760.3814922

Matthias Galster, Seyedmoein Mohsenimofidi, Jai Lal Lulla, Muhammad Auwal Abubakar, Christoph Treude, and Sebastian Baltes. 2026b. Configuring Agentic AI Coding Tools: An Exploratory Study. In *Proceedings of the 3rd ACM International Conference on AI-Powered Software (AIware '26).* Association for Computing Machinery, New York, NY, USA, 11–20. arXiv:2602.14690 doi:10.1145/3805760.3814887

Haoyu Gao, Jai Lal Lulla, Hong Yi Lin, Sebastian Baltes, Christoph Treude, and Mansooreh Zahedi. 2026b. From Registry to Repository: How AI Agent Skills Are Written, Adapted, and Maintained. arXiv:2607.00911 doi:10.48550/arXiv.2607.00911

Haoyu Gao, Jai Lal Lulla, Hong Yi Lin, Sebastian Baltes, Christoph Treude, and Mansooreh Zahedi. 2026c. From Registry to Repository: How AI Agent Skills Are Written, Adapted, and Maintained (Replication Package). doi:10.5281/zenodo.21032973

Yudong Gao, Zongjie Li, Yuanyuan Yuan, Zimo Ji, Pingchuan Ma, and Shuai Wang. 2026a. SkillReducer: Optimizing LLM Agent Skills for Token Efficiency. arXiv:2603.29919 doi:10.48550/arXiv.2603.29919

Naman Jain, Jaskirat Singh, Manish Shetty, Liang Zheng, Koushik Sen, and Ion Stoica. 2025. R2E-Gym: Procedural Environments and Hybrid Verifiers for Scaling Open-Weights SWE Agents. arXiv:2504.07164 doi:10.48550/arXiv.2504.07164 Published, COLM 2025.

Johannes Kirmayr, Lukas Stappen, and Elisabeth André. 2026. CAR-bench: Evaluating the Consistency and Limit-Awareness of LLM Agents under Real-World Uncertainty. arXiv:2601.22027 doi:10.48550/arXiv.2601.22027

Jin Rae Lee. 2026. *skill-invocation-cost-pilot.* Retrieved October 5, 2026 from https://github.com/theejer/skill-invocation-cost-pilot

George Ling, Shanshan Zhong, and Richard Huang. 2026. Agent Skills: A Data-Driven Analysis of Claude Skills for Extending Large Language Model Functionality. arXiv:2602.08004 doi:10.48550/arXiv.2602.08004

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the Middle: How Language Models Use Long Contexts. *Transactions of the Association for Computational Linguistics* 12 (2024), 157–173. doi:10.1162/tacl_a_00638

Jai Lal Lulla, Matthias Galster, Jie M. Zhang, Sebastian Baltes, and Christoph Treude. 2026. The Impact of Configuring Agentic AI Coding Tools on Build-vs-Buy Decisions: A Study Protocol. arXiv:2606.03907 doi:10.48550/arXiv.2606.03907 Accepted, ESEIW 2026 Registered Reports Track (ESEM).

OpenAI. 2026a. *API Pricing.* Retrieved October 5, 2026 from https://developers.openai.com/api/docs/pricing

OpenAI. 2026b. *codex — Model Catalog Source Code,* `models.json`. Retrieved October 10, 2026 from https://github.com/openai/codex/blob/092d3acd6bec3e3a14bdc7e7a2810ab628ab759d/codex-rs/models-manager/models.json

OpenAI. 2026c. *codex — Model Defaults Source Code,* `openai_models.rs`. Retrieved October 4, 2026 from https://github.com/openai/codex/blob/726f1492db90501746639e8c61cd3c674735a1a7/codex-rs/protocol/src/openai_models.rs

OpenAI. 2026d. *codex — Skill Catalog Prompt Source Code,* `catalog_prompt.rs`. Retrieved October 4, 2026 from https://github.com/openai/codex/blob/f3037cafd32b2097dae718eb86bcc9182dc0a5d1/codex-rs/ext/skills/src/catalog_prompt.rs

OpenAI. 2026e. *codex — Source Code,* `local_file_system.rs`. Retrieved September 1, 2026 from https://github.com/openai/codex/blob/3a04482645b695085f4daf7c6310ab8592653fea/codex-rs/exec-server/src/local_file_system.rs

OpenAI. 2026f. *Configuration Reference.* Retrieved October 2, 2026 from https://learn.chatgpt.com/docs/config-file/config-reference

OpenAI. 2026g. *GPT-6.1 Sol.* Retrieved October 5, 2026 from https://developers.openai.com/api/docs/models/gpt-6.1-sol

OpenAI. 2026h. *Reasoning Models.* Retrieved October 4, 2026 from https://developers.openai.com/api/docs/guides/reasoning

OpenAI. 2026i. *Using Skills to Accelerate OSS Maintenance.* Retrieved August 25, 2026 from https://developers.openai.com/blog/skills-agents-sdk

OpenCode. 2026a. *opencode — Model Request Source Code,* `session/llm/request.ts`. Retrieved October 2, 2026 from https://github.com/anomalyco/opencode/blob/e9f8a210b9e2b1e13d375b84906069886eb3b767/packages/opencode/src/session/llm/request.ts

OpenCode. 2026b. *opencode — Provider Request Options Source Code,* `provider/transform.ts`. Retrieved October 4, 2026 from https://github.com/anomalyco/opencode/blob/610df0b56674fa0ebcae89093ada08b98731bea6/packages/opencode/src/provider/transform.ts

OpenCode. 2026c. *opencode — Skill List Source Code,* `skill/index.ts`. Retrieved September 1, 2026 from https://github.com/anomalyco/opencode/blob/ebece6efd7b11401cf1e7390b5a22991b6608cc4/packages/opencode/src/skill/index.ts

OpenCode. 2026d. *OpenCode Documentation.* Retrieved August 22, 2026 from https://opencode.ai/docs/

OpenRouter. 2026. *OpenRouter: A Unified Interface for LLMs.* Retrieved August 12, 2026 from https://openrouter.ai/

Gergely Orosz and Elin Nilsson. 2026. *AI Tooling for Software Engineers in 2026.* Retrieved August 22, 2026 from https://newsletter.pragmaticengineer.com/p/ai-tooling-2026

Shuyin Ouyang, Jie M. Zhang, Mark Harman, and Meng Wang. 2025. An Empirical Study of the Non-Determinism of ChatGPT in Code Generation. *ACM Transactions on Software Engineering and Methodology* 34, 2, Article 42 (2025), 28 pages. doi:10.1145/3697010

Pi. 2026a. *Pi — a Minimal, Extensible Coding Agent.* Retrieved August 22, 2026 from https://pi.dev/

Pi. 2026b. *pi — Anthropic Request Source Code,* `api/anthropic-messages.ts`. Retrieved October 4, 2026 from https://github.com/earendil-works/pi/blob/b271b0a524b29e13c0c9e748aea0d34e1597f2db/packages/ai/src/api/anthropic-messages.ts

Pi. 2026c. *pi — CLI Documentation,* `docs/cli.md`. Retrieved October 2, 2026 from https://github.com/earendil-works/pi/blob/6f1072cc081f06b86a673bd142f03720d17afe15/packages/coding-agent/docs/cli.md

Pi. 2026d. *pi — OpenAI Responses Request Source Code,* `api/openai-responses.ts`. Retrieved October 4, 2026 from https://github.com/earendil-works/pi/blob/76dfb88f63ce51ff2e3fe2ead4fcf1f65f71f121/packages/ai/src/api/openai-responses.ts

Pi. 2026e. *pi — Skill Harness Source Code,* `harness/skills.ts`. Retrieved September 1, 2026 from https://github.com/earendil-works/pi/blob/b8b873b9872db04a938fb4357b5e8e824ddc051c/packages/agent/src/harness/skills.ts

Harsh Raj, Niranjan Orkat, Suvrorup Mukherjee, Aritra Guha, Cheryl Flynn, and Subhabrata Majumdar. 2026. Consistency as a Testable Property: Statistical Methods to Evaluate AI Agent Reliability. arXiv:2605.10516 doi:10.48550/arXiv.2605.10516

Vercel. 2026. *Agent Skills Directory.* Retrieved August 25, 2026 from https://skills.sh

Hironori Washizaki (Ed.). 2024. *Guide to the Software Engineering Body of Knowledge (SWEBOK Guide), Version 4.0.* IEEE Computer Society, Los Alamitos, CA. Retrieved August 3, 2026 from https://www.swebok.org

Abel Yagubyan. 2026. How Consistent Are LLM Agents? Measuring Behavioral Reproducibility in Multi-Step Tool-Calling Pipelines. arXiv:2605.28840 doi:10.48550/arXiv.2605.28840

Di Yang, Xinou Xie, Xiuwen Yang, Ming Hu, Yihao Huang, Yueling Zhang, Weikai Miao, Ting Su, Chengcheng Wan, and Geguang Pu. 2026. Clarity Is Not Assumed: Understanding LLM-Based Code Generation under Ambiguous Requirements. arXiv:2604.21505 doi:10.48550/arXiv.2604.21505

Jehyeok Yeon, Isha Chaudhary, and Gagandeep Singh. 2025. Quantitative Certification of Agentic Tool Selection. arXiv:2510.03992 doi:10.48550/arXiv.2510.03992

YanZhao Zheng, ZhenTao Zhang, Chao Ma, YuanQiang Yu, JiHuai Zhu, Yong Wu, Tianze Xu, Baohua Dong, Hangcheng Zhu, Ruohui Huang, and Gang Yu. 2026. SkillRouter: Skill Routing for LLM Agents at Scale. arXiv:2603.22455 doi:10.48550/arXiv.2603.22455 Preprint, under review.