SKILLIF: DO AGENTS FOLLOW THEIR SKILLS?

Anonymous authors
Paper under double-blind review

ABSTRACT

Agent skills specify what an agent should produce, how it should perform the work, and which limits it should respect. Evaluating their use therefore requires examining whether agents follow these requirements during execution. We introduce SkillIF, a benchmark for systematically studying adherence to skill specifications. SkillIF pairs 96 curated skills across ten domains with executable tasks and derives 2,194 assessable constraints from their specifications. A shared taxonomy enables comparisons of the same requirement types across different skills, while deterministic checks and LLM judgments assess each constraint against execution evidence. Together, these components reveal which requirements agents repeatedly fail to satisfy and how those failures appear in their actions and outputs. Across combinations of language models and execution harnesses, procedural requirements emerge as a shared weakness, while explicit prohibitions are followed more consistently than scope limits and stopping requirements. Substantial violations also remain in runs accepted by the task judge. Analysis of successful runs with low adherence distinguishes missing content and unexecuted steps from outputs that fulfill requirements only partly or in a different form. SkillIF provides a systematic account of how agents follow reusable specifications and identifies concrete targets for evaluating and improving skill adherence.

1 INTRODUCTION

Reusable skills provide agents with instructions for recurring kinds of work, combining workflows with references, templates, and scripts (Xu & Yan, 2026; Liang et al., 2026; Ling et al., 2026). They specify the intended output, the procedure for producing it, and the limits on execution. A skill may require an agent to consult a source, inspect a file before modifying it, or stop when a condition is met. Understanding whether skills guide agent behavior therefore requires checking whether agents follow these requirements as they work. Instruction-following benchmarks establish the importance of assessing explicit requirements (Zhou et al., 2023; Jiang et al., 2024; Qin et al., 2024), while agent evaluations study procedures, execution quality, and skill use (Qi et al., 2026; Li et al., 2025; Fan et al., 2026; Han et al., 2026; Zhu et al., 2026). Within this broader effort, we ask **which kinds of requirements within skills do agents satisfy, and which repeatedly remain unmet?** Answering this question requires comparing the same kinds of requirements across different workflows and retaining the evidence behind each judgment. For example, a requirement to include supporting information calls for different evidence from a requirement to consult a source before acting. Likewise, avoiding a prohibited operation does not establish that an agent respected scope limits or stopping conditions. Skill specifications and task requests must also remain distinct, so that an outcome accepted by the task judge can still be checked for unmet skill requirements. These considerations motivate a benchmark that supports comparisons across skills while tracing individual violations to execution evidence.

We introduce SkillIF to support this study (Figure 1). It pairs 96 curated skills across ten domains with executable tasks whose briefs state the assignment without reproducing the skill's procedure. Each agent receives the relevant skill, allowing evaluation to focus on adherence during execution. From the specifications, we derive 2,194 assessable constraints organized into **Deliverable**, **Procedure**, and **Restraint**, with nine subcategories. Deterministic checks and LLM judgments assess these constraints using recorded actions, responses, and artifacts. This framework supports comparisons across requirement types and analysis of how individual requirements go unmet. Across eleven combinations of models and execution harnesses, Procedure is consistently the lowest-scoring category, with evidence and step requirements particularly weak. Within Restraint, explicit prohibitions

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

are followed more consistently than scope limits and stopping requirements. These comparisons show which requirements account for weaknesses in overall adherence. Examining successful runs with low adherence then reveals how those weaknesses appear in execution. Required content or actions may be absent, while other outputs fulfill a requirement only partly or depart from its specified form. These findings identify concrete behaviors that future improvements need to address. Our contributions are:

- We introduce SkillIF, a benchmark for studying adherence to reusable skill specifications. Its 96 skill–task pairs keep the assignment distinct from the skill's requirements, enabling separate assessments of task completion and adherence on the same execution.
- We develop a framework that organizes 2,194 assessable constraints under a shared requirement taxonomy and connects each judgment to execution evidence. It enables comparisons across heterogeneous skills and diagnosis of how individual requirements remain unmet.
- We characterize recurring weaknesses across eleven model–harness configurations, including procedural requirements and differences among behavioral limits. Analysis of successful runs with low adherence further identifies missing content, unexecuted actions, and partial or nonconforming outputs as concrete targets for improvement.

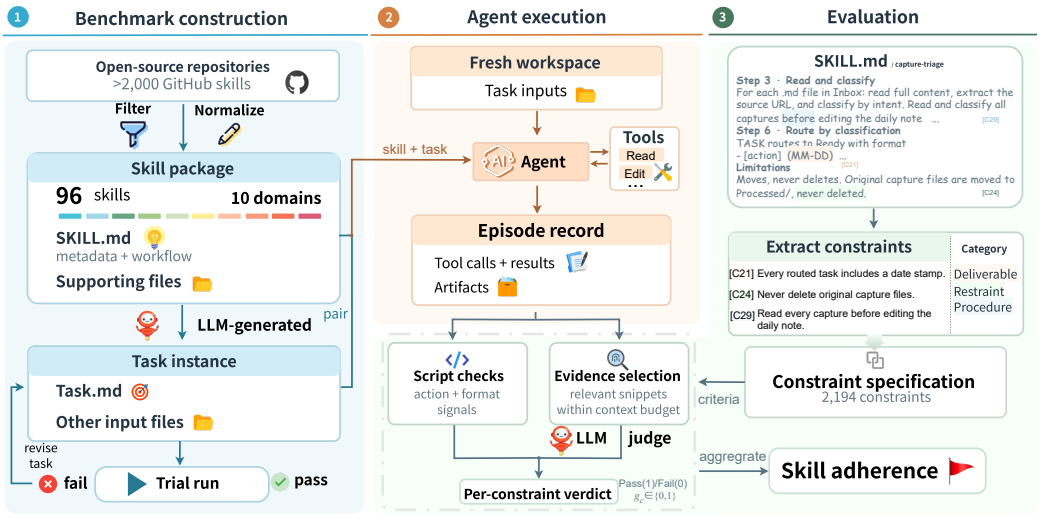


Figure 1: Overview of SkillIF. Skills are paired with tasks, and their requirements are converted into categorized constraints. Deterministic checks and LLM judgments assess each constraint against execution evidence, enabling overall adherence scores, comparisons across requirement types, and analysis of specific violations.

2 Related Work

Agents and executable tasks. Agents combine reasoning, environment interaction, and tool use (Yao et al., 2022; Schick et al., 2023). Benchmarks assess task completion across software development, web interaction, and other executable environments (Jimenez et al., 2024; Zhou et al., 2024; Liu et al., 2024b; Trivedi et al., 2024; Merrill et al., 2026). SkillIF examines whether agents follow the skill specifications that guide their execution. **Instruction following and constraint satisfaction.** Instruction-following benchmarks evaluate explicit requirements through verifiable constraints, controlled difficulty, and decomposed assessment (Zhou et al., 2023; Jiang et al., 2024; Qin et al., 2024), extending to multilingual and multi-turn settings, constraint composition, and system-level instructions (He et al., 2024; Wen et al., 2024; Qin et al., 2025). Agentic evaluations cover long instructions and file-based workflows (Qi et al., 2026; Chen et al., 2026a), policy and procedural adherence (Yao et al., 2024; Li et al., 2025), and intermediate execution quality and detailed requirements (Fan et al., 2026; Li et al., 2026a; Zhuge et al., 2024). SkillIF derives the requirements it assesses from reusable skill packages, covering outputs, action sequences, and behavioral limits.

Agent skills and skill-use evaluation. Research develops reusable code skills (Wang et al., 2023), instruction-and-resource packages and skill infrastructure (Xu & Yan, 2026; Liang et al., 2026; Li et al., 2026b), and methods for discovering, revising, evaluating, and verifying skills (Zhang et al., 2026b; Yang et al., 2026; Zhong et al., 2026; Ma et al., 2026; Gautam et al., 2026). SkillsBench studies performance gains from supplied skills (Li et al., 2026c); Skill-Use examines whether agents appropriately use skills during tasks (Han et al., 2026); and SkillCoach uses process evaluation to support training (Zhu et al., 2026). Related work examines logical relations within specifications (Chen et al., 2026b), skill–task matching (Liu et al., 2026), and execution differences with and without skills (Zhou et al., 2026). To compare adherence across skills, individual judgments must be organized by the kinds of requirements they assess. SkillIF groups requirements from heterogeneous skill packages into common types, connects judgments to execution evidence, and identifies how unmet requirements appear in actions and outputs. This organization enables comparisons of the same requirement type across skills and reveals differences within broad categories, such as prohibitions, scope limits, and stopping conditions, while retaining the evidence behind each verdict.

3 SKILLIF BENCHMARK

3.1 PROBLEM FORMULATION

We study an agent carrying out a concrete task with a supplied skill. An evaluation instance pairs a skill $S = (D, R)$, where D is the instruction document and R its supporting resources, with a task $T = (q, X)$, where q is the task request and X the input materials. The task defines the particular assignment; the skill provides reusable requirements for performing that kind of work. We supply this pairing directly, leaving skill retrieval outside the evaluation. An agent configuration $A = (M, H)$ combines a language model M with an execution harness H, the software environment through which the agent accesses instructions and tools. It produces an execution record τ containing tool calls and their results, the final response, and workspace artifacts. Our primary target, **skill adherence**, measures the extent to which τ satisfies the requirements C_s extracted from skill S, where s indexes the supplied skill. We judge each requirement individually and aggregate the verdicts as defined in Section 3.4. We assess task success separately, allowing runs judged successful to be examined for unmet requirements on outputs, procedures, and behavioral limits.

3.2 SKILL COLLECTION AND TASK CONSTRUCTION

To distinguish adherence to a skill from fulfillment of the immediate request, our tasks specify a concrete goal while leaving the workflow and its requirements in the skill. SkillIF contains 96 skills drawn from more than 2,000 candidates collected from GitHub repositories (Appendix I). Selection proceeded in two stages. A model first screened the candidates for skills that support substantive work, retaining 355. We then trial-ran each skill's own workflow and discarded 259 whose environments could not be reproduced in our evaluation setting or whose execution exceeded practical runtime limits. Before deriving constraints, we normalize each retained skill so that corresponding sections follow a common structure across the collection. Normalization standardizes document structure and converts tabular rules into self-contained sentences while preserving the requirements. Appendix B provides the normalization specification and rewriting prompts. The resulting package contains SKILL.md and supporting resources, such as references, templates, and scripts. The header identifies the skill and its intended use; the body specifies its workflow and requirements. The collection spans ten application domains, with 8–12 skills per domain, covering software development and design, business operations, and professional services such as law, finance, and compliance.

The normalized packages provide the basis for both task construction and adherence evaluation. For each skill, an LLM receives the complete package and generates a task brief and accompanying input files. The brief specifies a concrete assignment and expected deliverable without naming the skill or reproducing its procedure. This keeps the immediate request distinct from the reusable instructions for carrying it out. Each generated task then undergoes a separate trial run to verify that its inputs are available and the assignment can be executed in the evaluation environment. Tasks that fail this check are revised and tested again. This second trial checks task executability; adherence is assessed separately against the skill's requirements. After the task is finalized, we derive a fixed constraint set from the normalized skill specification, as described in Section 3.3, and use it to assess every execution of that skill.

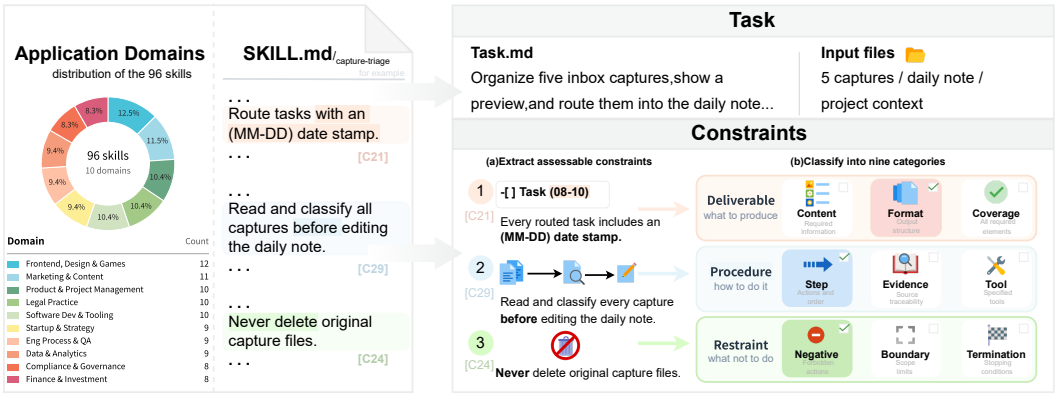


Figure 2: **The benchmark at a glance.** The application domains summarize the 96 skills. The capture-triage example shows a SKILL.md beside the task generated from it and the constraints drawn from it. Each marked requirement becomes a single assessable constraint, which is then classified into one of the nine subcategories of Table 8.

3.3 Constraint Specification

A constraint expresses a requirement from the normalized skill in a form that can be judged individually against execution evidence. We preserve what the skill asks the agent to do while making the requirements comparable across workflows. Constraints are organized by what they govern into three categories and nine subcategories. **Deliverable** covers the output: *content* specifies required information, *format* specifies its structure or encoding, and *coverage* requires all specified elements to be included. **Procedure** covers how work is performed: *step* specifies actions and their order, *evidence* requires claims to be grounded in sources, and *tool* specifies tool invocation. **Restraint** covers behavioral limits: *negative* forbids particular actions, *boundary* restricts scope, and *termination* covers when execution must stop, whether as a condition on the run or as a stop rule the deliverable is required to state. Classification depends on what a requirement asks for, rather than where it appears or which evidence is used to judge it. For example, reading every capture before editing a daily note is a step constraint, adding an MM-DD date stamp is a format constraint, and never deleting original captures is a negative constraint (Figure 2). This taxonomy lets us compare the same kinds of requirements across skills and distinguish the weaknesses behind low adherence scores.

We derive constraints by first segmenting each skill document into candidate instruction units. During this initial stage, headings and code blocks are set aside, prose is divided into sentences, and enumerations are expanded into separate items. A model assigns each unit a subcategory or excludes material that states no behavioral requirement, such as background or terminology. A rule-based pass then splits compound requirements only where the parts can be judged separately: lists and coordinated verbs become individual conditions, whereas quantified, sequenced, and conditional requirements stay whole, so one constraint can still carry several obligations. A final pass examines each candidate alongside the complete package, specifies the evidence needed to judge it, and associates a deterministic check where available. Supporting scripts and schemas clarify existing requirements without introducing obligations absent from the specification. We then remove semantically duplicate requirements within each skill and details too minor to assess independently, retaining 2,194 of 3,284 candidates (Appendix A.9). The resulting constraints connect output requirements to produced files, procedural requirements to recorded actions, and preservation rules to operations and resulting artifacts. Of these constraints, 1,918 have deterministic checks and 276 rely on LLM judgment alone. Each constraint specification records its condition, subcategory, and evaluation mode. Section 3.4 explains how the evidence is used to determine its verdict.

3.4 Evaluation Protocol

Constraint judgments. Each requirement is assessed using the relevant parts of the execution record: tool calls and their results, the final response, and workspace artifacts. Our evaluation

builds on LLM-as-a-judge and rubric-based approaches (Zheng et al., 2023; Zhuge et al., 2024; Rao & Callison-Burch, 2026; Ding, 2026). The adherence criteria themselves come from the supplied skill specification, so each judgment concerns a stated requirement and the evidence relevant to it. Deterministic checks test mechanically verifiable properties, such as file content, tool-call order, or forbidden operations. An LLM judge assesses whether the requirement is satisfied in substance. When a deterministic check is available, the constraint passes only if both the check and the LLM judge accept it. Otherwise, the LLM judge alone determines the verdict. Each assessed constraint receives a binary verdict: 1 if it is judged satisfied under this rule, and 0 otherwise.

A failed deterministic check means that the available evidence did not pass the encoded test; what this indicates depends on the property being tested. For requirements naming an element the deliverable must contain, such as a label, a format, or a required stop rule, a mismatch can fail the check even when the content is semantically similar. Action-based checks instead test whether an operation occurred, occurred in the required order, or was avoided. Under the joint acceptance rule, failure of either the deterministic check or the LLM assessment results in a failed verdict. The failure analysis distinguishes missing content and actions from partial fulfillment and deviations from specified forms (Section 4.4).

Adherence scores. For a fixed model–harness configuration, let C_s be the constraint set for skill s, and let $\widehat{C}_s \subseteq C_s$ contain the constraints for which a judgment is returned. A small number receive no verdict and are excluded from both the numerator and denominator throughout. With $g_{s,c} \in \{0, 1\}$ denoting the verdict on constraint c, instance-level and overall adherence are

$$\mathrm{SA}_s = \frac{1}{|\widehat{C}_s|} \sum_{c \in \widehat{C}_s} g_{s,c}, \qquad \mathrm{Overall} = \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} \mathrm{SA}_s,$$

where $\mathcal{S}$ is the benchmark skill set. Macro-averaging gives each skill equal weight despite differences in constraint counts.

We apply the same principle to subcategory and category scores. For a subcategory or category x, let $\widehat{C}_{s,x}$ contain the judged constraints of skill s that belong to x, and let $\mathcal{S}_x$ contain the skills for which this set is nonempty. The reported score is

$$A_x = \frac{1}{|\mathcal{S}_x|} \sum_{s \in \mathcal{S}_x} \frac{1}{|\widehat{C}_{s,x}|} \sum_{c \in \widehat{C}_{s,x}} g_{s,c}.$$

These scores measure adherence to particular requirement types. Their unweighted average need not equal Overall because both the constraint counts and the contributing skill sets can differ across types. All adherence scores are reported as percentages.

Auxiliary task success. A separate LLM judgment assesses whether the requested outcome was achieved, using a task-derived rubric and the execution evidence without access to the skill specification. Task success is the proportion of runs accepted by this judge and does not enter the adherence score. It provides an outcome-based comparison for examining which skill requirements remain unmet in runs judged to have completed the task.

4 Experiments

4.1 Experimental Setup

We evaluate six models under Claude Code and five of the same models under Codex, yielding eleven model–harness configurations (Table 1). The five shared models enable within-model harness comparisons, motivated by prior work on agent interfaces and execution environments (Yang et al., 2024; Yao et al., 2026; Zhang et al., 2026a). Each configuration is evaluated on the same 96 skill–task pairs using a fixed set of 2,194 constraints. We score one execution for each pair under each configuration, yielding 1,056 runs in total. Every execution starts in a fresh workspace and is instructed to load the paired skill before acting. The agent first sees the skill's name and description, then accesses its full instructions through the harness. Skill adherence is the primary metric; task success is an auxiliary measure of outcome completion, as defined in Section 3.4. DeepSeek V4 Flash at temperature zero provides both LLM judgments through separate prompts. We report scores

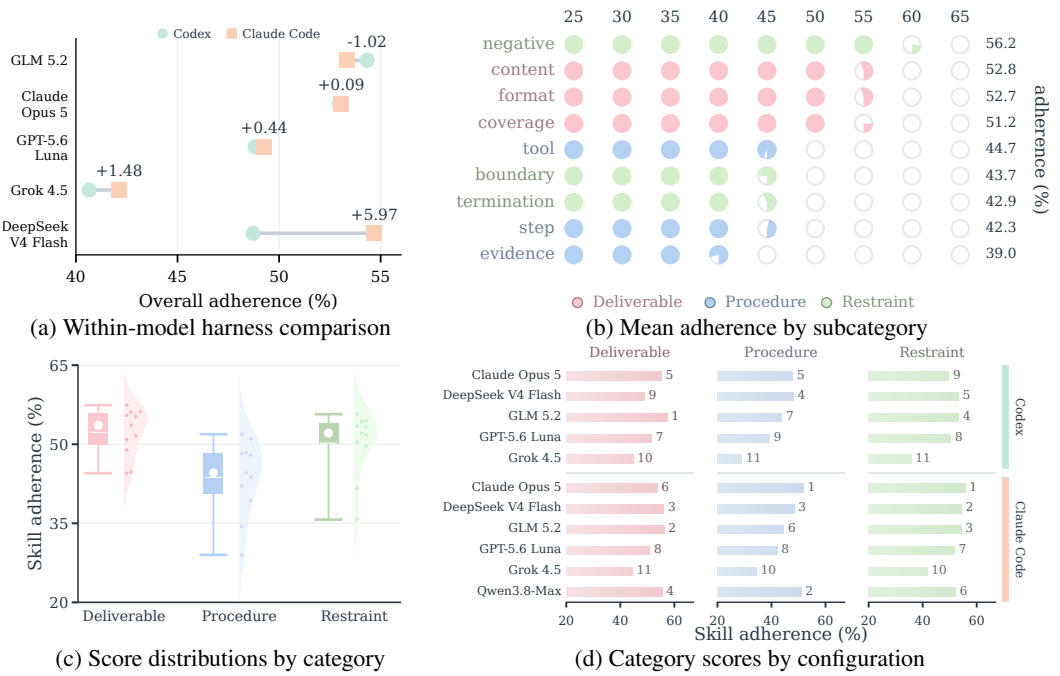
from the first of three independent judging passes over the same execution records. Appendix A provides the execution settings and the judging protocol.

Judge sensitivity. Re-evaluating the same records with a second judge model, while leaving the rest of the pipeline unchanged, lowers mean Overall adherence by 2.65 percentage points. The set of six highest-scoring configurations remains unchanged, although their internal ordering shifts. The second judge also preserves 48 of the 55 pairwise orderings between configurations, the same number the primary judge preserves between its own first and third passes. Absolute scores are therefore sensitive to judge choice, while the broad configuration ordering is largely preserved (Appendix A.6).

Table 1: **Skill adherence (%) by model, harness, and requirement type.**

	Claude Opus 5		DeepSeek V4 Flash		GLM 5.2		GPT-5.6 Luna		Grok 4.5		Qwen 3.8-Max
	Codex	Claude Code	Codex	Claude Code	Codex	Claude Code	Codex	Claude Code	Codex	Claude Code	Claude Code
Deliverable	55.3	53.6	48.9	56.1	**57.4**	**56.2**	51.6	50.9	44.7	44.5	55.5
Content	56.3	54.1	50.1	55.9	**60.2**	56.8	52.0	51.5	43.2	42.9	**57.7**
Format	**58.1**	55.7	50.8	**62.1**	52.4	58.3	50.3	46.2	48.6	43.3	54.2
Coverage	50.2	53.8	**62.4**	52.5	59.4	52.7	50.9	48.2	34.9	41.5	**57.1**
Procedure	47.9	**51.9**	**48.2**	48.4	43.8	44.6	39.3	42.1	29.0	34.4	51.0
Step	42.0	**52.2**	**49.3**	44.2	42.1	43.1	37.4	39.2	31.0	37.1	47.7
Evidence	**50.1**	**50.7**	35.7	46.0	45.5	38.9	35.8	39.8	20.0	17.7	48.3
Tool	**44.4**	47.5	43.9	55.1	**44.4**	45.0	35.4	49.5	34.3	35.4	**56.6**
Restraint	49.7	**55.7**	53.3	54.4	**53.4**	54.3	50.4	51.8	35.7	41.7	52.1
Negative	56.9	60.7	**60.6**	**62.2**	58.4	56.0	56.6	59.0	44.1	45.9	57.4
Boundary	44.0	43.6	43.5	48.6	**50.1**	**52.3**	49.4	48.2	19.3	32.6	49.4
Termination	42.7	**54.4**	**48.5**	45.6	42.7	52.9	38.2	30.9	30.9	39.7	45.6
Overall	**53.0**	**53.1**	**48.7**	**54.7**	**54.3**	**53.3**	**48.8**	**49.3**	**40.7**	**42.1**	**54.0**

Scores are macro-averaged over skills with assessed requirements of the corresponding type. Bold indicates the best score per row within each harness, except Overall, which is bold throughout. Qwen3.8-Max is evaluated only under Claude Code.



(a) Within-model harness comparison

(b) Mean adherence by subcategory

(c) Score distributions by category

(d) Category scores by configuration

Figure 3: **Skill adherence across harnesses and requirement types.** (a) Within-model harness comparisons; labels show Claude Code − Codex in percentage points. (b) Mean subcategory scores across eleven configurations. (c) Category score distributions across configurations; white dots mark means. (d) Category scores grouped by harness; labels indicate ranks among all eleven configurations within each category.

4.2 Adherence across Skill Requirements

Overall adherence ranges from 40.7% to 54.7% across the eleven configurations (Table 1). Breaking these scores down by requirement type reveals a more consistent pattern than the overall rankings: agents struggle most with procedural requirements, and adherence to behavioral limits varies with the kind of limit imposed.

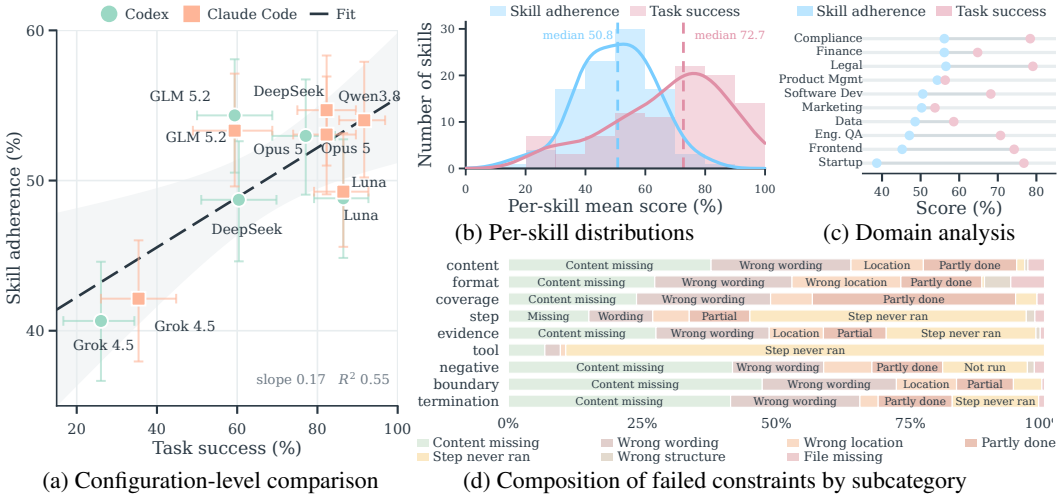
Procedures are a shared weakness. Procedure is the lowest-scoring category in all eleven configurations. Averaged across configurations, each of its three subcategories scores below every Deliverable subcategory, with evidence and step requirements especially weak (Figure 3(b,c)). The weakness thus extends beyond invoking specified tools to consulting sources and carrying out prescribed actions. The configurations leading on Deliverable and Procedure also differ: more reliable outputs do not imply more reliable adherence to the process. Comparisons within the same skill show that this pattern is not simply due to procedural requirements appearing in harder assignments. Evidence and step requirements pass less often than other requirements in the same skill, by 7.9 and 6.9 percentage points, respectively, in the specification-property analysis over the six Claude Code configurations (Appendix F.3); both differences keep their sign on the Codex configurations. These comparisons hold the skill fixed, although individual requirements can still differ in complexity and assessment method. The weakness also persists when the same records are judged repeatedly. Procedure scores below Deliverable in all 33 comparisons across configurations and judging passes, and is the lowest category in 32 (Appendix A.5). Across these analyses, agents follow requirements on the result more consistently than requirements on the work needed to produce it.

Prohibitions, scope, and stopping differ. Every configuration follows negative requirements more consistently than boundary or termination requirements (Figure 3(b,d)). Restraint therefore captures distinct aspects of adherence rather than a uniform ability to respect limits. Avoiding a specified action, staying within an assigned scope, and specifying when work must stop impose different requirements and can call for different evidence. For example, a prohibition may be satisfied by the absence of an action, whereas a requirement to state a stopping rule in a deliverable must be checked in that deliverable. Both belong to Restraint because of the limits they specify; the evidence used to judge them does not determine their category. A skill may forbid deleting files while separately limiting which files may be modified or when further action is permitted. Avoiding deletion alone does not establish that these other limits were respected. Boundary requirements also pass less often than other requirements within the same skill, by 7.5 percentage points, and keep the same sign on the Codex configurations (Appendix F.3). Their weakness is therefore not explained solely by which skills contribute to the aggregate score.

The harness changes a model's adherence profile. Alongside these shared weaknesses, adherence varies with the execution configuration. DeepSeek V4 Flash scores higher under Claude Code, whereas Claude Opus 5 changes little in Overall but gains on Procedure and Restraint while losing on Deliverable (Figure 3(a,d)). Changing the harness can therefore alter a model's ranking and the kinds of requirements it satisfies, even when its overall score barely changes. The average difference between harnesses depends on which models are included and does not indicate a uniform benefit (Figure 6). Configurations with similar scores can exchange ranks across judging passes, while Procedure remains a common weakness. These comparisons describe the evaluated executions; identifying which harness components account for the differences would require controlled changes to the execution environment.

4.3 Task Success Does Not Imply Skill Adherence

A run can pass the task evaluation while leaving many skill requirements unmet. Of 717 runs accepted by the task judge, 298 (41.6%) satisfy fewer than half of their assessed skill constraints. The discrepancy occurs within the same executions: their outputs meet the task rubric, but many requirements governing how those outputs are prepared or presented remain unmet. In the go-to-market example, the agent produces the requested plan and passes all seven task criteria, yet satisfies only two of sixteen skill constraints (Table 11; Appendix C). The unmet requirements include analysis before drafting the plan and verification before presenting it. The outcome-based task evaluation can accept the plan without establishing that the prescribed work took place. This makes task success a useful comparison for examining skill adherence in runs whose immediate outcomes are accepted.



(a) Configuration-level comparison (b) Per-skill distributions (c) Domain analysis (d) Composition of failed constraints by subcategory

Figure 4: **Task success, skill adherence, and failure forms.** (a) Configuration-level scores with 95% bootstrap intervals over skills; the dashed regression line has a nominal 95% confidence band. (b) Per-skill score distributions, averaged across eleven configurations. (c) Domain means across skills and configurations. (d) Failure-form proportions within each requirement subcategory, based on 4,347 failed judgments from 298 task-successful runs with adherence below 50%. Four of the six models are abbreviated in (a): Opus 5 is Claude Opus 5, DeepSeek is DeepSeek V4 Flash, Luna is GPT-5.6 Luna, and Qwen3.8 is Qwen3.8-Max.

Stronger task performance also does not determine a configuration's adherence profile. The two measures are positively associated across configurations, but their rankings differ. Under Codex, GPT-5.6 Luna leads on task success while GLM 5.2 leads on adherence (Tables 1 and 9). Within individual skills, the relationship is weaker, with a mean correlation of 0.25 across the 92 skills for which it is defined. Task completion and skill adherence are thus related at the configuration level, but provide different evidence about execution. In SkillIF, task success is an auxiliary measure and is not included in the adherence score. The next analysis focuses on accepted runs to identify violations that remain when the immediate assignment is judged complete.

4.4 How Successful Runs Depart from Skill Specifications

The 298 successful runs with adherence below 50% contain 4,347 failed constraint judgments, each assessing one requirement in one execution. To characterize these failures, we assign each failed judgment one of seven failure forms based on the preceding evidence assessment (Appendix E). A manual review of 600 sampled judgments, with the original verdicts hidden, agrees with 89.9% of the 348 judgments labeled as failures and 95.2% of the 252 labeled as passes. Agreement on the assigned failure form is 87.4% among the sampled failed judgments (Appendix A.7). The failure analysis describes the selected subset of successful runs with low adherence, rather than all benchmark failures.

Required work may be missing, incomplete, or presented differently. Content missing and step never ran account for 29.9% and 23.5% of failures, respectively, when the nine requirement subcategories are weighted equally (Table 2). We calculate each failure form's proportion within each subcategory and then average the nine proportions, preventing subcategories with more failed judgments from dominating the summary. Table 2 pairs each failure form with a stated requirement and the corresponding execution evidence. Requirement subcategories describe what the skill asks for; failure forms describe how the execution falls short. A single subcategory can therefore contain several failure forms (Figure 4(d)); Appendix E.3 illustrates both labels for one restraint requirement. Appendix Tables 12 and 13 provide the weighting formula and compare this summary with one that weights each failed judgment equally. The weighting changes the proportions, but both summaries retain the distinction between requirements that are never fulfilled and outputs that are produced but do not meet the requirement.

Table 2: **The seven assigned forms, their share, and one example of each.** Shares are the unweighted mean of the nine subcategories' own compositions over the 4,347 failed checks of the 298 selected runs. Requirements and execution evidence are condensed; Appendix Table 16 gives the examples in full.

Form assigned	Macro-averaged share	What the skill required	What the run did
Required content, actions, or files absent			
Content missing	29.9%	The report header includes a `WCAG Version` field.	The header carries no version field; the string appears nowhere in the output.
Step never ran	23.5%	Read the bundled method document under `resources/` before writing the failure narrative.	The tool log shows reads of two other bundled resources and none of this one.
File missing	1.9%	Create the local settings file, at the project root under `file/setting`.	The file is never created.
Produced but not as required			
Wrong wording	19.9%	The same-day fixes form a tier labelled *Quick Wins*.	The tier exists, but is labelled *CLASS A — Fixable immediately*; the required name appears nowhere.
Partly done	14.6%	Monitoring covers CPU, memory, disk and network.	Only disk appears, and only in a log-rotation context; the other three are absent from the monitoring layer.
Wrong location	9.2%	The plan contains a `### Sprint Goal` section.	The goal is stated in the chat reply only; the plan body has no such section.
Wrong structure	1.0%	The output opens with the `## Outreach Sequence` heading.	The heading is present, but process narration precedes it: *I'll review the outreach workflow … before drafting*.

Failure forms clarify the weak requirements. Missing actions are particularly common among failed step and tool requirements: step never ran accounts for 51.5% and 89.3% of their failures, respectively. Poor procedural adherence therefore includes required operations with no recorded execution. Failures of evidence requirements are less concentrated in a single form, with unexecuted steps and missing content occurring at similar rates. Coverage failures often involve partial completion, as in the monitoring example. These forms describe the available evidence; they do not explain the agent's internal reasoning.

Some requirements must be checked during execution. A final review may correct missing content or an incorrectly structured output. It cannot undo a violation of required timing or order: if inspection must precede modification, inspecting afterwards does not make the earlier modification compliant. Improvement therefore depends on when a requirement needs to guide the agent's decisions, as well as whether the final artifact can be repaired. Some failures call for revisions to the deliverable; others require checks before the agent takes the next action. This distinction identifies concrete targets for improvement, including missing required elements, incomplete or differently expressed outputs, and actions whose conditions must be respected during execution. A supplementary experiment on 38 skills examines edits to instruction placement and repetition. It yields an estimated gain of 1.66 percentage points (95% interval -3.1 to 6.5; one execution per condition; Appendix F.4), leaving the effectiveness of these edits unresolved.

5 Conclusion

SkillIF asks whether agents follow their skills. Across the evaluated configurations, the answer is only partly: agents repeatedly miss requirements governing procedures and behavioral limits. SkillIF makes these weaknesses measurable by deriving individual constraints from skill specifications, organizing them under a shared taxonomy, and assessing them against execution evidence. Procedure is the lowest-scoring category in all eleven configurations, while scope and stopping requirements are followed less consistently than explicit prohibitions. Substantial nonadherence also remains in runs accepted by the task judge, with 41.6% satisfying fewer than half of their assessed requirements. Examining the execution evidence distinguishes missing content and unexecuted actions from partial or nonconforming outputs. These distinctions identify concrete targets for improvement and clarify which requirements must be checked during execution. Together, the benchmark, evaluation framework, and empirical findings provide a systematic account of how agents fulfill and violate reusable skill specifications. SkillIF makes adherence to those specifications a concrete, testable target for agent development.

AI USE STATEMENT

Generative AI was used in benchmark construction, evaluation, and manuscript preparation. LLMs assisted with skill screening and normalization, as well as requirement classification. All task briefs and accompanying input materials were generated entirely by AI and subsequently tested for executability. Automated evaluation also relied on LLM judgments, as detailed in the methods and appendices. The authors directed the research and retained responsibility for its design, analysis, and interpretation. Extracted constraints were checked against their source skills, and samples of automated judgments were independently reviewed with the original verdicts withheld. AI tools also assisted with polishing the text of the manuscript; the authors reviewed and approved the final text, results, tables, and references.

REPRODUCIBILITY STATEMENT

We document the benchmark construction and evaluation procedures in Section 3, with execution settings and evidence-processing details in Appendix A. Appendix B describes skill normalization and provides the associated prompts. The appendices further specify constraint selection, score aggregation, and manual verification procedures. Each skill–configuration pair is evaluated through one agent execution; repeated judging passes assess evaluation variability on the same execution records. Reproducing the reported results requires the evaluated skill packages, task briefs and input files, constraint specifications, evaluation code, and the model and harness settings described in the paper. Because execution and evaluation involve language models, independently repeated runs may yield different trajectories and scores.

REFERENCES

Kaiyuan Chen, Qimin Wu, Taiyu Hou, Tianhao Tang, Xueyu Hu, Yuchen Hou, Bikun Li, Chengming Qian, Guoyin Wang, Haolin Chen, et al. Agentif-oneday: A task-level instruction-following benchmark for general ai agents in daily scenarios. *arXiv preprint arXiv:2601.20613*, 2026a.

Xuan Chen, Chengpeng Wang, Lu Yan, and Xiangyu Zhang. Slbench: Evaluating how llm agents follow logical relations in skills. *arXiv preprint arXiv:2607.09016*, 2026b.

Kaustubh Deshpande, Ved Sirdeshmukh, Johannes Baptist Mols, Lifeng Jin, Ed-Yeremai Hernandez-Cardona, Dean Lee, Jeremy Kritz, Willow E Primack, Summer Yue, and Chen Xing. Multichallenge: A realistic multi-turn conversation evaluation benchmark challenging to frontier llms. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 18632–18702, 2025.

Liang Ding. Adarubric: Task-adaptive rubrics for reliable llm agent evaluation and reward learning. *arXiv preprint arXiv:2603.21362*, 2026.

Shengda Fan, Xuyan Ye, Yupeng Huo, Zhi-Yuan Chen, Yiju Guo, Shenzhi Yang, Wenkai Yang, Shuqi Ye, Jingwen Chen, Haotian Chen, et al. Agentprocessbench: Diagnosing step-level process quality in tool-using agents. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pp. 8823–8834, 2026.

Srishti Gautam, Arjun Radhakrishna, and Sumit Gulwani. Skillaxe: Sharpening llm-authored agent skills through evaluation-guided self-refinement. *arXiv preprint arXiv:2606.10546*, 2026.

Jinyi Han, Yuanjian Xu, Ying Liao, Xinyi Wang, Zishang Jiang, Zixiang Di, Fanyang Lu, Zhichao Hu, and Yanghua Xiao. Skill-use: Can llms actually use skills in agentic harnesses? *arXiv preprint arXiv:2608.04828*, 2026.

Yun He, Di Jin, Chaoqi Wang, Chloe Bi, Karishma Mandyam, Hejia Zhang, Chen Zhu, Ning Li, Tengyu Xu, Hongjiang Lv, et al. Multi-if: Benchmarking llms on multi-turn and multilingual instructions following. *arXiv preprint arXiv:2410.15553*, 2024.

Yuxin Jiang, Yufei Wang, Xingshan Zeng, Wanjun Zhong, Liangyou Li, Fei Mi, Lifeng Shang, Xin Jiang, Qun Liu, and Wei Wang. Followbench: A multi-level fine-grained constraints following benchmark for large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4667–4688, 2024.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, volume 2024, pp. 54107–54157, 2024.

Dawei Li, Yuguang Yao, Zhen Tan, Huan Liu, and Ruocheng Guo. Toolprmbench: Evaluating and advancing process reward models for tool-using agents. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 12378–12391, 2026a.

Hao Li, Chunjiang Mu, Jianhao Chen, Siyue Ren, Zhiyao Cui, Yiqun Zhang, Lei Bai, and Shuyue Hu. Organizing, orchestrating, and benchmarking agent skills at ecosystem scale. *arXiv preprint arXiv:2603.02176*, 2026b.

Xiangyi Li, Yimin Liu, Wenbo Chen, Bingran You, Zonglin Di, Yifeng He, Shenghan Zheng, Kyoung Whan Choe, Jiankai Sun, Shuyi Wang, et al. Skillsbench: Benchmarking how well agent skills work across diverse tasks. *arXiv preprint arXiv:2602.12670*, 2026c.

Zekun Li, Shinda Huang, Jiangtian Wang, Nathan Zhang, Antonis Antoniades, Wenyue Hua, Kaijie Zhu, Sirui Zeng, Chi Wang, William Yang Wang, et al. Sopbench: Evaluating language agents at following standard operating procedures and constraints. *arXiv preprint arXiv:2503.08669*, 2025.

Yuan Liang, Ruobin Zhong, Haoming Xu, Chen Jiang, Yi Zhong, Runnan Fang, Jia-Chen Gu, Shumin Deng, Yunzhi Yao, Mengru Wang, et al. Skillnet: Create, evaluate, and connect ai skills. *arXiv preprint arXiv:2603.04448*, 2026.

George Ling, Shanshan Zhong, and Richard Huang. Agent skills: A data-driven analysis of claude skills for extending large language model functionality. *arXiv preprint arXiv:2602.08004*, 2026.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the association for computational linguistics*, 12:157–173, 2024a.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. Agentbench: Evaluating llms as agents. In *International Conference on Learning Representations*, volume 2024, pp. 52989–53046, 2024b.

Yujian Liu, Jiabao Ji, Li An, Tommi Jaakkola, Yang Zhang, and Shiyu Chang. How well do agentic skills work in the wild: Benchmarking llm skill usage in realistic settings. *arXiv preprint arXiv:2604.04323*, 2026.

Yuchen Ma, Yue Huang, Han Bao, Haomin Zhuang, Swadheen Shukla, Michel Galley, Xiangliang Zhang, and Stefan Feuerriegel. Skillgen: Verified inference-time agent skill synthesis. *arXiv preprint arXiv:2605.10999*, 2026.

Mike Merrill, Alexander Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Shin, Thomas Walshe, E Kelly Buchanan, et al. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *International Conference on Learning Representations*, volume 2026, pp. 40903–40986, 2026.

Valentina Pyatkin, Saumya Malik, Victoria Graf, Hamish Ivison, Shengyi Huang, Pradeep Dasigi, Nathan Lambert, and Hanna Hajishirzi. Generalizing verifiable instruction following. *Advances in Neural Information Processing Systems*, 38, 2026.

Yunjia Qi, Hao Peng, Xiaozhi Wang, Amy Xin, Youfeng Liu, Bin Xu, Lei Hou, and Juanzi Li. Agentif: Benchmarking large language models instruction following ability in agentic scenarios. *Advances in Neural Information Processing Systems*, 38, 2026.

Yanzhao Qin, Tao Zhang, Yanjun Shen, Wenjing Luo, Yan Zhang, Yujing Qiao, Weipeng Chen, Zenan Zhou, Wentao Zhang, and Bin CUI. Sysbench: Can llms follow system message? In *International Conference on Learning Representations*, volume 2025, pp. 74442–74466, 2025.

Yiwei Qin, Kaiqiang Song, Yebowen Hu, Wenlin Yao, Sangwoo Cho, Xiaoyang Wang, Xuansheng Wu, Fei Liu, Pengfei Liu, and Dong Yu. Infobench: Evaluating instruction following ability in large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 13025–13048, 2024.

Delip Rao and Chris Callison-Burch. Autorubric: A unifying framework for rubric-based llm evaluation on non-verifiable tasks. In *Third Conference on Language Modeling*, 2026.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551, 2023.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16022–16076, 2024.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

Bosi Wen, Pei Ke, Xiaotao Gu, Lindong Wu, Hao Huang, Jinfeng Zhou, Wenchuang Li, Binxin Hu, Wendy Gao, Jiaxin Xu, et al. Benchmarking complex instruction-following with multiple constraints composition. *Advances in Neural Information Processing Systems*, 37:137610–137645, 2024.

Renjun Xu and Yang Yan. Agent skills for large language models: Architecture, acquisition, security, and the path forward. *arXiv preprint arXiv:2602.12430*, 2026.

John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

Yutao Yang, Junsong Li, Qianjun Pan, Bihao Zhan, Yuxuan Cai, Lin Du, Jie Zhou, Kai Chen, Qin Chen, Xin Li, et al. Autoskill: Experience-driven lifelong learning via skill self-evolution. *arXiv preprint arXiv:2603.01145*, 2026.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

Yilun Yao, Xinyu Tan, Chao-Hsuan Liu, Yaoming Li, Zhengyang Wang, Wenhan Yu, Zhewen Tan, Yuxuan Tian, Guangxiang Zhao, Lin Sun, et al. Harness-bench: Measuring harness effects across models in realistic agent workflows. *arXiv preprint arXiv:2605.27922*, 2026.

Tao Zhang, Chenglin Zhu, Yanjun Shen, Wenjing Luo, Yan Zhang, Hao Liang, Fan Yang, Mingan Lin, Yujing Qiao, Weipeng Chen, et al. Cfbench: A comprehensive constraints-following benchmark for llms. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 32926–32944, 2025.

Yunbei Zhang, Janet Wang, Yingqiang Ge, Weijie Xu, Jihun Hamm, and Chandan K Reddy. Stop comparing llm agents without disclosing the harness. *arXiv preprint arXiv:2605.23950*, 2026a.

Ziao Zhang, Kou Shi, Shiting Huang, Avery Nie, Yu Zeng, Yiming Zhao, Zhen Fang, Qishen Su, Haibo Qiu, Wei Yang, et al. Skillflow: Benchmarking lifelong skill discovery and evolution for autonomous agents. *arXiv preprint arXiv:2604.17308*, 2026b.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.

Shanshan Zhong, Yi Lu, Jingjie Ning, Yibing Wan, Lihan Feng, Yuyi Ao, Leonardo FR Ribeiro, Markus Dreyer, Sean Ammirati, and Chenyan Xiong. Skilllearnbench: Benchmarking continual learning methods for agent skill generation on real-world tasks. *arXiv preprint arXiv:2604.20087*, 2026.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, volume 2024, pp. 15585–15606, 2024.

Xiaolin Zhou, Jinbo Liu, Li Li, Ryan A Rossi, and Xiyang Hu. Counterfactual trace auditing of llm agent skills. *arXiv preprint arXiv:2605.11946*, 2026.

Jiayin Zhu, Kelong Mao, Yudong Guo, Dengbo He, Sulong Xu, Simiu Gu, and Yutao Yue. Skillcoach: Self-evolving rubrics for evaluating and enhancing agentic skill-use. *arXiv preprint arXiv:2607.01874*, 2026.

Mingchen Zhuge, Changsheng Zhao, Dylan Ashley, Wenyi Wang, Dmitrii Khizbullin, Yunyang Xiong, Zechun Liu, Ernie Chang, Raghuraman Krishnamoorthi, Yuandong Tian, et al. Agent-as-a-judge: Evaluate agents with agents. *arXiv preprint arXiv:2410.10934*, 2024.

A Execution and Evaluation Details

This appendix specifies the execution conditions and evidence available to the evaluators. Appendix B documents the normalization applied to each skill before constraints were derived. Appendix C expands the comparison between task success and adherence, and Appendix D reports the per-domain averages. Appendix E examines the failures within successful runs. Appendices F and G then distinguish associations with specification properties, results of targeted edits, and the checks used to interpret those findings. Appendix H compares SkillIF with representative benchmarks, restricted to the features each reported evaluation protocol makes explicit. Appendix I lists the repositories the candidate skills were collected from.

A.1 Configurations and Skill Access

The evaluation includes six models under the Claude Code CLI—Claude Opus 5, Qwen3.8-Max, DeepSeek V4 Flash, GLM 5.2, GPT-5.6 Luna, and Grok 4.5—and the same set except Qwen3.8-Max under the Codex CLI. All eleven configurations receive the same 96 skill–task pairs, task briefs, and input materials. We retain one scored execution for each pair and configuration, giving 1,056 runs. The five shared models support matched harness comparisons; Qwen contributes to configuration-level results but not to the matched harness average. Model names identify the models served during evaluation. The runs were served through provider endpoints rather than pinned builds, so the exact build behind a name is not recoverable from it alone.

Every run starts in a fresh workspace and is instructed to load the paired skill before taking any other action. The invocation instruction is supplied separately from the task brief, which neither names the skill nor reproduces its procedure. Both harnesses disclose skills progressively: names and descriptions are initially available, and full instructions are accessed through the skill interface. Codex may shorten descriptions when its skill-description budget is exceeded. The experiment therefore holds skill selection fixed and measures execution against an identified relevant skill, so that a score reflects how a skill was followed rather than whether it was found.

A.2 Execution Limits and Repeated Runs

Each execution has a 3,000-second wall-clock limit and a single agent attempt; the longest recorded run takes 2,148 seconds. The evaluation setup imposes no additional cap on tool calls, turns, tokens,

or cost, and no run records a budget hit. Context management remains with the respective CLI rather than an additional truncation, retrieval, or summarization layer in the benchmark runner. Workspace names use millisecond timestamps, but these are identifiers rather than sampling seeds; no sampling seed is passed to either CLI.

The single-run design controls the task and input materials across configurations but does not estimate run-to-run variability. A matched comparison consequently describes performance on the recorded executions, and a bootstrap over skills measures sensitivity to the skill sample rather than variation from repeating an identical execution. This distinction also applies to the specification-edit experiment: retaining the model and rubric makes the before–after scores comparable, but does not eliminate stochastic variation between runs.

A.3 Evidence Processing and Constraint Judgments

Evaluation uses tool calls and results, the final response, and workspace artifacts. Both LLM assessments use DeepSeek V4 Flash at temperature zero, without an ensemble or sampling seed. The adherence judge processes batches of ten constraints, with up to three attempts and a 600-second timeout per call. A deterministic check, when available, runs under a 60-second timeout. Such a constraint passes only when both the check and the LLM judge accept it; constraints without a deterministic check use the LLM verdict. Requirements for which no verdict is returned are excluded from the score numerator and denominator as specified in Section 3.4. A missing verdict is therefore distinct from a failed constraint.

The judges assess the record a run leaves: the specification, the tool calls it made, the files it wrote, and the final response. That record is assembled by one fixed procedure for every run, so every configuration is judged on the same evidence and a judgment describes what the record shows. A verdict of absence is read the same way: an element recorded as missing is missing from that record.

Task success uses a separate task-derived rubric and does not receive the skill specification. The two prompts thus assess different obligations on the same execution: whether the requested outcome is achieved and whether the skill requirements are satisfied. Because both use the same judge model, separate prompts do not imply statistically independent errors. Their comparison is informative about the distinction between the two rubrics.

A.4 Composition of the Judging Surface

Adherence is assessed by two channels. Of the 2,194 constraints, 1,918 (87.4%) pair a deterministic script check with an LLM assessment and 276 (12.6%) rest on the LLM alone; every paired constraint requires both channels to accept it, and none is decided disjunctively. Across the 24,134 constraint judgments the evaluation produces, the script check already rejects 26.5%, leaving 73.5% to be settled by whether the LLM endorses an execution the check admits. Where the check passes, the LLM withholds its endorsement in 29.5% of cases. The balance is not uniform across subcategories: 90.1% of negative-constraint judgments depend on the LLM, against 50.5% for tool and 53.1% for step. The deterministic channel therefore fixes the upper part of the score range, and the distinctions this paper draws are largely made within what remains.

Two checks bound how far a requirement can be satisfied by being restated. Among 1,643 constraints whose required value could be extracted as a literal string, 808 (49.2%) also occur verbatim in the skill's own SKILL.md, 149 (9.1%) occur only in the task brief, and 686 (41.8%) occur in neither. The first pattern does not predict adherence: a required value present in SKILL.md and absent from the task brief carries an odds ratio of 1.010 ($p = 0.92$) in the specification-property analysis of Appendix F. The second does: a required value that appears in the task brief is met 12.1 points more often under matching (odds ratio 1.496, $[1.257, 1.780]$). Reproducing a specification in the output therefore earns no credit by itself, whereas a brief that discloses a required value makes that requirement easier to satisfy. Both patterns concern where a value is stated rather than whether the agent complied, and neither is a property of the agent.

The adherence judge is blind to the identity of the run it assesses. Its prompt contains the specification text, the final response, and the evidence blocks assembled for the requirement, and the tool log exposes only a tool's name, input, and result; no field carries a model, harness, or configuration label. Self-preference risk is therefore confined to whatever a writing style reveals rather than to an

explicit identifier. The judge is DeepSeek V4 Flash, itself one of the eleven configurations evaluated here under both harnesses, so two of the eleven are graded by the model that executed them. Two checks bound what that can carry: re-judging with a substitute drawn from outside the family leaves the tier structure unchanged (Appendix A.6), and a manual re-read of 348 verdicts agrees with the reported judge on 89.9% of calls and reaches $\kappa = 0.84$ on the assigned form (Appendix A.7). The two checks bound the objection in its two directions—how far a score moves with the judge, and how far the judge moves from a human reading—which is why the configuration comparisons are reported as judgments rather than as measurements.

A.5 Repeated Judging and Verdict Stability

The adherence verdicts reported throughout this paper come from the first of three independent re-judge passes over the same eleven configurations and 96 skills. Re-judging recomputes each deterministic script check from the transcript and independently obtains LLM judgments for all constraints, applying the same combination rule as in Section 3.4; task success is not re-judged and therefore has no counterpart to the checks below. The second and third passes serve as a stability check and supply no score reported elsewhere in this paper.

Script checks are reproducible by construction, and were: all 31,636 re-judged check verdicts are identical across the three passes. LLM verdicts are a fresh judgment each pass and agree across all three passes in 20,270 of 23,791 cases (85.2%). The 3,521 disagreements (14.8%) are spread across configurations rather than concentrated in one, with per-configuration agreement between 82.5% and 89.1%. Neither channel changes its coverage between passes: the set of judged criteria is the same in all three, so these rates are taken over the full judged set and not over a subset common to the passes. Nineteen of the 1,056 configuration–skill cells are left out of these counts because their verdicts are carried from the archive rather than re-judged, so they are identical across passes by construction; they are the 19 cells of the two skills for which no fresh verdict was obtained. Counting them moves LLM agreement to 85.4%. Table 4 relates these counts to one another.

At the level at which this paper reports, the passes are close. Averaged over the eleven configurations, Overall moves from 50.2% to 50.6% across the three passes, a range of 0.4 points, and the three category means move by at most 0.8 points. Procedure, the bottleneck identified in Section 4.2, moves by 0.08 points. A single configuration's Overall moves by at most 1.7 points and a single category cell by at most 5.0, the larger movements falling on Restraint, the category carrying the fewest constraints. Agreement is lower on quantities the paper reports as secondary checks: the mean within-skill correlation between task success and adherence is 0.25 in the reported pass, against 0.27 and 0.28 in the other two.

Table 3 gives each configuration's Overall under each pass. The judge's contribution to a configuration's score is therefore small relative to the differences the paper reports, even though it is large enough to unsettle a comparison between neighbouring rows.

The disagreement rate translates into a magnitude. Taking the per-skill difference between two passes gives a root-mean-square of 8.8 points, corresponding to a standard deviation of about 6.2 points for a single skill under a single pass. Averaged over 96 skills, one pass places a configuration's mean with a standard error of roughly 0.6 points, and the difference between two configurations with roughly 0.9; averaging three passes reduces these to about 0.4 and 0.5. Comparing the three-pass average against the first two passes alone moves every configuration by at most 0.4 points.

The passes separate a stable finding from an unstable one. The procedural bottleneck is the stable one: Procedure falls below Deliverable in all eleven configurations in every pass, and is the lowest of the three categories in 32 of the 33 configuration–pass cells. The identity of a leader is not: the gap between neighbouring configurations is often smaller than the movement a pass introduces, the two leading configurations of Table 1 are 0.35 points apart, and the model leading a harness block is not the same in all three passes. Statements that turn on which of two close configurations ranks higher therefore describe the reported pass rather than a stable property of the benchmark.

The size of the gap makes the same point quantitatively. Under the three-pass average the three leading configurations lie within 0.30 points of one another, against a standard error of about 0.5 points on the difference between two of them, so the spacing is under one standard error. The leading configurations are consequently better read as a group than as an order.

Table 3: **Overall adherence under each re-judging pass.** Each column is an independent pass over the same eleven configurations and 96 skills. Pass 1 supplies the scores reported in Table 1.

Harness	Model	Pass 1	Pass 2	Pass 3
Codex	Claude Opus 5	53.0	53.5	54.2
	GLM 5.2	54.3	54.4	55.6
	DeepSeek V4 Flash	48.7	50.1	49.8
	GPT-5.6 Luna	48.8	48.2	49.4
	Grok 4.5	40.7	40.5	39.6
Claude Code	Qwen3.8-Max	54.0	54.8	55.5
	DeepSeek V4 Flash	54.7	55.2	55.3
	GLM 5.2	53.3	52.9	53.6
	Claude Opus 5	53.1	53.6	53.5
	GPT-5.6 Luna	49.3	50.9	49.2
	Grok 4.5	42.1	41.7	41.3
Mean over configurations		**50.2**	**50.5**	**50.6**

Table 4: **The counts behind the judging surface, and how each is obtained.** Where a value has two numbers the first is the script channel and the second the LLM channel.

Quantity	Value	How it is obtained
Configuration–skill cells	1,056	11 configurations $\times$ 96 skills
Constraints in the taxonomy	2,194	Table 8
Constraint–configuration units	24,134	one per constraint, per cell
Stored channel keys	32,042 / 24,090	every check a skill ships; the constraint units that returned a verdict
Carried from the archive	406 / 299	the two skills whose verdicts are carried, 19 cells
Re-judged	31,636 / 23,791	32,042 $-$ 406 and 24,090 $-$ 299
Identical across the three passes	31,636	all re-judged script verdicts
Unanimous across the three passes	20,270	85.2% of the re-judged LLM verdicts
Both judges gave a verdict	23,735	23,749 $-$ 14; 23,749 is 24,090 $-$ 341, the units in the 22 cells either judge carried from the archive

A.6 Substituting the Judge Model

Every score reported in this paper comes from one judge model, DeepSeek V4 Flash, leaving open the objection that the results describe the judge rather than the executions it assessed. The judge and the executor are separate roles throughout: the harness and model columns of the tables name the executor that produced a run, whereas the judge is the model that graded it. One model fills both roles, since DeepSeek V4 Flash executed runs as well as judging them, and the substituted judge Qwen3.8-Flash is a different model from the executor Qwen3.8-Max. We tested the objection by re-judging all eleven configurations and 96 skills with a second judge, Qwen3.8-Flash, holding everything else fixed: the same execution records, the same 2,194 constraints, the same batches of ten, the same evidence assembly, and temperature zero. Only the judge model changed. No score reported elsewhere in this paper uses these verdicts.

The substitution moves every configuration in the same direction. Mean Overall falls from 50.2% to 47.5%, a shift of 2.65 points, and all eleven configurations decline; the smallest movement is 0.5 points and the largest 5.5, so the shift is uniform in sign but not in size. Table 5 gives each configuration. The eleven movements share a judge pair rather than being independent draws, so the uniform direction is read as a description of the shift rather than as a test of it.

Changing the judge moves scores more than repeating the judge does. Across the three re-judge passes of Appendix A.5, a configuration's Overall moves by at most 1.7 points and the configuration means span 50.2% to 50.6%; substituting the judge moves the mean by 2.65 points and every con-

Table 5: **Overall adherence under a substituted judge.** Rows are execution configurations and the two scored columns are the judge that graded their runs, so a heading that repeats a row label means one model served in both roles. The first scored column is the reported pass (Table 3, Pass 1); the second re-judges the same execution records with Qwen3.8-Flash under an otherwise identical pipeline. The shift is the first column minus the second, taken before rounding, so it can differ by 0.1 from the difference of the printed columns. Every row declines, and the six leading configurations are the same six under both columns, though their order within it is not.

Executor		Judge		
Harness	**Model**	**DeepSeek V4 Flash**	**Qwen3.8-Flash**	**Shift**
Codex	Claude Opus 5	53.0	50.0	2.9
	GLM 5.2	54.3	48.8	5.5
	DeepSeek V4 Flash	48.7	46.1	2.7
	GPT-5.6 Luna	48.8	45.9	2.9
	Grok 4.5	40.7	39.0	1.7
Claude Code	Qwen3.8-Max	54.0	53.5	0.5
	DeepSeek V4 Flash	54.7	51.7	3.0
	GLM 5.2	53.3	50.9	2.4
	Claude Opus 5	53.1	50.7	2.3
	GPT-5.6 Luna	49.3	47.3	2.0
	Grok 4.5	42.1	38.9	3.2
Mean over configurations		**50.2**	**47.5**	**2.6**

figuration in the same direction. The identity of the judge therefore matters for the level at which a configuration scores.

It matters less for the ordering, and no more than the judge's own repeatability does. Under the substitution 48 of the 55 pairs of configurations keep their relative order, a Kendall τ of $+0.745$; the reported judge's own first and third passes give the same number, 48 of 55, and the same $+0.745$. Against a spread of about 14 points between the strongest and weakest configurations, reordering of that size is not specific to the change of judge. The tiers are unchanged: the six highest configurations under the substitute are the same six as under the reported judge, and the two Grok 4.5 configurations are weakest under both. The order inside the leading group is not, and one configuration falls four places.

The shift is a change of judgment rather than a change of coverage. Outside the cells either judge carried from the archive the reported judge scores 23,749 constraint units and the substituted judge 23,735 of them, so 14 are scored by the reported judge alone; restoring its verdicts for those changes the mean by 0.01 points. The remainder is disagreement on evidence both judges saw: of the 23,735 constraints both assessed, 3,119 (13.1%) receive different verdicts, and the disagreement is asymmetric, with the substituted judge stricter on 1,974 and more lenient on 1,145, a ratio of 1.7 to 1. The asymmetry is spread across the taxonomy rather than concentrated in one subcategory, running from 1.2 to 1 on format to 2.6 to 1 on coverage (Table 6).

The substitution bounds the sensitivity of the reported scores to the identity of the judge; it is not a ranking of the two. A uniformly stricter judge is consistent with two models reading the same requirement at different levels of literalness, and what the reported scores rest on is the reported judge's own agreement with a manual re-read: 89.9% on the call and $\kappa = 0.84$ on the assigned form (Appendix A.7).

A.7 Agreement with a Manual Re-read

The adherence verdicts throughout this paper are assigned automatically, and a sample of them was re-read by hand. 348 judgments were re-read against the requirement text and the record of the run that the assignment had seen, with the assigned verdict withheld, so the check is of the assignment and not of the retrieval that assembled the record; each re-read produced an independent

Table 6: **Disagreement between the two judges by requirement subcategory.** The reported judge is DeepSeek V4 Flash, which also appears among the executors of Table 1 and therefore graded some runs it had produced; the substitute is Qwen3.8-Flash, a different model from the executor Qwen3.8-Max. Each row covers the constraints both judges scored in cells both judges re-judged, pooling all eleven configurations and 96 skills. *Stricter* counts the constraints the reported judge passed and the substituted judge failed; *more lenient* counts the reverse. The ratio is stricter to more lenient. Subcategories follow Table 8.

Category	Subcategory	Judged	Stricter	More lenient	Ratio
Deliverable	content	10,951	812	470	1.7
	format	3,385	228	190	1.2
	coverage	1,647	154	60	2.6
Procedure	step	2,342	269	152	1.8
	evidence	1,363	108	54	2.0
	tool	517	34	26	1.3
Restraint	negative	2,023	248	119	2.1
	boundary	1,089	91	56	1.6
	termination	418	30	18	1.7
All		**23,735**	**1,974**	**1,145**	**1.7**

call and, wherever it too placed the requirement in the unmet class, one of the seven failure forms of Appendix E. The sample follows the evidence each assignment rests on: a deterministic script check failed the requirement in 150 of the judgments, a script check passed and an LLM judgment failed it in 130, and an LLM judgment alone failed it in 68.

The 600 judgments were drawn before the re-read under one fixed seed and one set of rules, from two populations: the runs the paper analyses, which supply the 348 unmet judgments and 152 of the met ones, and the runs whose adherence the score clears, which supply the remaining 100. Within a stratum, runs were drawn at equal probability; within a drawn run, at most five items of that stratum were taken and a run entered at most two strata, so no run contributed more than ten judgments. The drawn items were renumbered after shuffling so that a judgment's stratum cannot be read off its identifier, and machine paths were scrubbed from the worksheets. Two re-readers worked independently, neither seeing the assigned verdict or the other's worksheet, and a third re-reader resolved the judgments on which the two calls differed; that resolution is the re-read reported here.

Table 7 reports the comparison, with both columns taken over the same 348 judgments. The re-read and the assignment agree on the call in 313 of them (89.9%, 95% Wilson interval $[86.3, 92.7]$). They agree on the form in 304 (87.4%, $[83.5, 90.4]$), counting an agreement only where the re-read also placed the requirement in the unmet class and named the same form there. The form rate is necessarily the lower of the two, because the 35 judgments the re-read took out of that class cannot carry a matching form. Of the 44 disagreements, 35 are on the call and 9 are judgments the two placed in the same class but assigned different forms. Agreement on the call is highest where a script check decided the assignment (93.3%) and lowest in the stratum resting on an LLM judgment alone (85.3%), the one without a mechanical anchor; the form column keeps the same order (90.7%, 87.7%, 79.4%).

Two features of these rates are properties of the design rather than of the re-read. The form column carries the chance-corrected agreement. Both raters vary across the seven forms, and taking the 35 judgments the re-read took out of the unmet class as a category of their own, observed agreement is 0.874 against an expected 0.218, a κ of 0.84, against 38.8% for a re-read that always named the most common form. The call column counts movements instead. Every judgment in the sample was assigned the unmet class, so a re-read that answered the same way throughout would score 100%, and the assigned call is constant by construction in the two strata a script check decided, which leaves the column no chance correction to make; what it records is the 35 judgments (10.1%) the re-read took out of that class.

Table 7: **Agreement between the assigned verdicts and a manual re-read.** The re-read saw the requirement and the record of the run, with the assigned verdict withheld. It returned a call on 597 of the 600 judgments and a failure form wherever it too placed the requirement in the unmet class. The upper block covers the 348 the assignment placed in the unmet class, and both of its columns are taken over those same 348: a judgment counts toward the form column only when the re-read also placed the requirement in the unmet class and assigned the same form there, so every disagreement on the call is a disagreement on the form as well. The lower block covers the 252 it placed in the met class, where the assignment has no form to compare against and the two form columns are empty. Intervals are 95% Wilson. *Script check failed*: a deterministic check decided the assignment. *Veto*: a script check passed and an LLM judgment failed the requirement. *LLM only*: no script check applied. The lower block pools two sampling frames.

		Call		Form	
How the judgment was reached	n	**Agreed**	**95% CI**	**Agreed**	**95% CI**
Script check failed	150	140 (93.3%)	[88.2, 96.3]	136 (90.7%)	[84.9, 94.4]
Veto	130	115 (88.5%)	[81.8, 92.9]	114 (87.7%)	[80.9, 92.3]
LLM only	68	58 (85.3%)	[75.0, 91.8]	54 (79.4%)	[68.4, 87.3]
Unmet class, all	**348**	**313 (89.9%)**	**[86.3, 92.7]**	**304 (87.4%)**	**[83.5, 90.4]**
Met class, all	**252**	**240 (95.2%)**	**[91.9, 97.3]**	—	—

Nine further judgments agree on the call and differ on the form. Five of the nine cross the distinction drawn in Section 4.4 between work that was never produced and work produced but not as required; the other four move between two forms on the same side of it, twice from missing content to a step that never ran. Those nine are a check on the assignment's internal consistency rather than a second measurement of it, and they locate the disagreement at a boundary the paper already draws rather than scattering it across the seven forms.

The re-read covered the other direction as well. A further 252 judgments that the assignment placed in the met class were re-read under the same protocol and the same blinding, 152 of them drawn from runs whose adherence falls below the cut the score reports against and 100 from runs it clears. The re-read took 9 out of the met class (3.6%, 95% Wilson interval $[1.9, 6.6]$), left 2 unsettled for want of evidence in the record, and found 1 inapplicable to its run; it confirmed the other 240 (95.2%, $[91.9, 97.3]$). The two directions are not symmetric, and here too the rate counts movements rather than a corrected level: every judgment in this arm was assigned the met class, so a re-read answering met throughout would score 100%, and what the 3.6% counts is the 9 taken out of that class—against the 35 of 348 (10.1%) taken out of the unmet class. The arm is not a simple random sample of all passing judgments: 188 of the 252 rest on a mechanical leg and 64 on an LLM judgment alone, and the narrower group carries a single disagreement, too few to order the two against each other.

A.8 Sensitivity to the Skill Sample

The scores above average over 96 skills, and the results could in principle rest on a few of them. Splitting the skills at random into two halves and scoring every configuration on each half, repeated 4,000 times, correlates the two halves' configuration vectors at 0.835; the Spearman–Brown correction for the halved length raises this to 0.910 for the full set. The two half-leaderboards rank the configurations at a Spearman correlation of 0.699, with a mean maximum displacement of 4.8 places. The two arbitrary halves therefore agree at 0.835, and the corrected estimate for the full skill set is 0.910, which is the sense in which the patterns reported here are properties of the skill set rather than of particular skills. Dropping any single skill leaves every configuration within two places of its full-set rank, and 291 of the 2,194 constraints (13.3%) return the same verdict under all eleven configurations, holding every configuration to the same requirements rather than separating them.

Subsampling gives a direct estimate of how many skills the two central findings require. Drawing skills at random, 500 subsets at each size, Procedure remains the lowest-scoring category once the eleven configurations' category means are pooled in 88% of twenty-skill subsets, 99% of fifty-skill

subsets, and every subset of sixty or more; the median number of configurations for which it is the lowest of the three categories rises from eight of eleven at twenty skills to ten at sixty. The gap between task success and adherence stays within five points of its full-set value in 74% of twenty-skill subsets, 98% of fifty-skill subsets, and every subset of sixty or more. Both findings are therefore carried by a random majority of the skills rather than by a few of them, and degrade only where the sample is small enough for individual skills to dominate the aggregate.

A.9 Constraint Selection

Derivation is deliberately over-inclusive. A skill's specification restates the same procedural action in more than one place, and it spells out details—individual formulas, boundary cases, evidence conventions—that are too minor to be worth scoring on their own. A constraint set built from all of it would count some requirements twice, pad the remainder with material nothing turns on, and end up longer than the requirements it is meant to represent. Selection therefore removes two kinds of candidate: those that duplicate a semantic action already covered by another candidate from the same skill, and those that record a detail of the second kind.

Of 3,284 candidates, 1,090 were removed and 2,194 retained; Table 8 reports the counts for each subcategory. Removal follows the two grounds above and is applied within each skill: a candidate goes when it restates a semantic action another candidate from the same skill already covers, or when it records a detail too minor to assess on its own.

The curation is one-sided. Candidates are not dropped for being hard: a candidate that no configuration satisfies is kept, because it records a requirement the specification imposes and dropping it would report a score over a shorter list than the skill states. The judgments also cover requirements that most configurations fail: of the 2,174 retained candidates that these judgments cover in all eleven configurations, 301 (13.8%) are failed in six of the eleven. They are unevenly distributed: failed in six of the eleven in 29.8% of tool candidates, 22.7% of step, 18.5% of coverage and of evidence, and 5.8% of negative. Section A.8 gives the related count.

A.10 Constraint Taxonomy and Aggregation

Table 8 gives the complete taxonomy. Of the 2,194 retained constraints, Deliverable accounts for 1,478, Procedure for 390, and Restraint for 326. The nine subcategories partition the constraints, and each subcategory belongs to exactly one category.

Adherence is first averaged within each skill and then across skills. Subcategory and category scores follow the same principle, including only skills with assessed constraints of the relevant type. This prevents skills with many constraints from dominating the overall score. It also means that category means need not average to the Overall row: the numbers of requirements and contributing skills differ. In contrast, the failure-form analysis pools failed judgments and normalizes within each subcategory. The property analysis likewise reports criterion-level pass rates and within-skill estimates. These quantities answer different questions and should not be read as interchangeable adherence scores.

A.11 Figure Details

The configuration panels average over the 96 skills. The intervals in Figure 4(a) are percentile bootstrap intervals over skills: each dot is placed at that configuration's observed means, 2,000 resamples of the 96 skills are drawn, and the horizontal bar is the interval on task success while the vertical bar is the interval on adherence. Both bars on a dot come from the same resample, so the two are paired rather than independent. The horizontal bars are the wider of the two, spanning 5.2 to 10.4 points against 3.8 to 4.4 vertically, because task success is close to binary at the skill level, where reclassifying a single skill moves a configuration. The shade around the dashed fit is a nominal 95% band on the least-squares line; unlike the bars it treats the eleven configurations as independent observations although five models contribute twice. Panel (b) draws kernel densities scaled to the histogram counts. Panel (c) averages each domain's skills over the eleven configurations, which counts the five models run under both harnesses twice. In panel (d) each subcategory's bar is normalized to that subcategory's own failed constraints, so the bar lengths compare proportions rather than counts. Domains differ in size and in requirement composition, so the rows describe heterogeneity rather than

Table 8: **Constraint taxonomy.** Nine subcategories across 96 skills, organized by what they constrain. *Candidates* counts the constraints derived from each skill's specification before selection, *Retained* those that enter the benchmark, and *Dropped* the difference (Appendix A.9).

Category	Constraint subcategory	Constraint	Candidates	Dropped	Retained
Deliverable	content	Information the output must contain	1,532	520	1,012
	format	Output structure and encoding	470	156	314
	coverage	Inclusion of all required elements	214	62	152
Procedure	step	Actions and their execution order	330	113	217
	evidence	Traceability of claims to sources	153	27	126
	tool	Invocation of a specified tool	124	77	47
Restraint	negative	Actions that must not be taken	287	99	188
	boundary	Limits on the agent's scope	122	22	100
	termination	Conditions under which to stop	52	14	38
Total			**3,284**	**1,090**	**2,194**

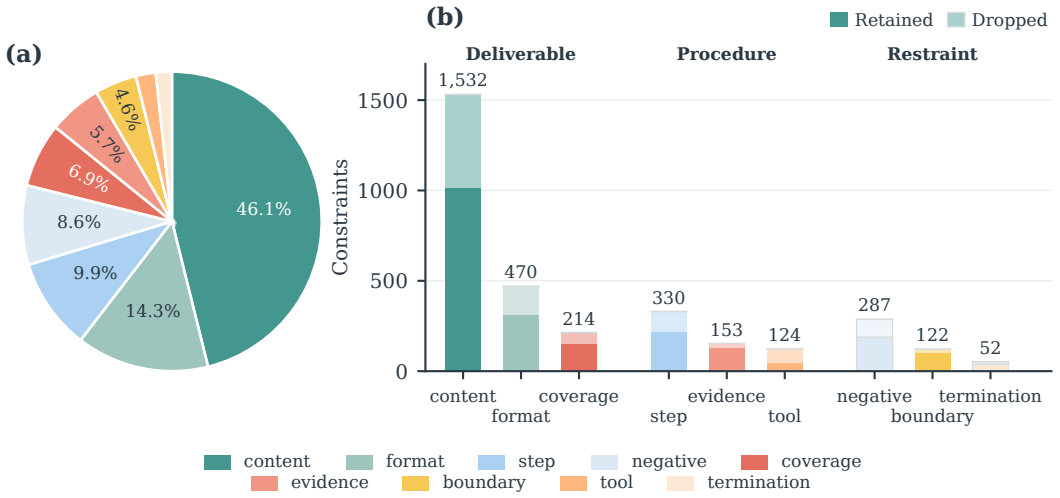


Figure 5: **Where the constraint pool sits, how much of it selection kept, and what the kept set is made of.** The nine subcategories of Table 8 in two views, side by side and keyed once beneath them; each keeps one color across both. (a) The 2,194 constraints that survived selection, as nine shares of that whole. Seven of its nine wedges print their share inside the disc and the other two are named only in the key. *content* alone is 46.1% of everything retained, more than the Procedure and Restraint families together (716) and 3.2× the next-largest, *format* at 14.3%. (b) The same nine as columns, in the table's own row order and grouped into its three families, each split into what selection retained (solid, at the base) and dropped (pale, above it), with the candidate count printed over it. A column's height is that subcategory's candidate count — 1,532 for *content* against 52 for *termination* — and its solid fraction the retention rate, 37.9% for *tool* to 82.4% for *evidence*. Pool size and how hard a pool is cut do not move together: the four subcategories nearest the pooled 66.8% span 287 to 1,532 candidates.

a ranking of domains by difficulty. Figure 3(b) and (c) average over the eleven configurations, so the five models run under both harnesses contribute twice, and the scores in both figures aggregate Table 1.

B SKILL NORMALIZATION

Each retained skill was normalized before any constraints were derived from it, under a written specification that predates the benchmark runs. The specification fixes the body structure of a skill and the form of its header description.

B.1 REQUIRED BODY SECTIONS

Every skill body must carry the following three sections, in any order and under any of the headings listed for it.

- Workflow or process: states what the skill does, step by step. Headings in the collection include Workflow, Execution Flow, and Process.
- Output format: states what the user receives and what the result looks like. Headings in the collection include Output, Output Format, and Deliverables.
- Scope or limitations: states what the skill does not do and when it should not be used. Headings in the collection include Scope, Limitations, and What This Skill Does NOT Do.

B.2 HEADER DESCRIPTION

The description follows a fixed pattern: it states in the third person what the skill does, followed by a clause naming the situations in which the skill applies. Imperative, first-person, and second-person openings are excluded. Cross-skill routing, of the form "not for X, use Y instead", is also excluded from the description and belongs in the scope section instead.

B.3 TABLE NORMALIZATION

Because constraints are derived by segmenting a document into sentences, material that is not already in sentence form is rewritten first. Tabular rules and thresholds are restated as self-contained sentences by a language model, under the following prompt.

> You are normalizing markdown tables in a skill definition (SKILL.md) into natural-language bullet lists. The downstream instruction-splitter splits text on line breaks, so tables get shredded into meaningless cells. Convert each table so its meaning is preserved as self-contained sentences.
>
> RULES:
>
> - Preserve ALL information: do not drop rows, columns, or meaning.
> - Output bullet items ("- ..."), one per original data row (or per unit of meaning).
> - Rewrite constraint-like tables (steps, rules, do/don't, priorities) as imperative or declarative constraint sentences.
> - Rewrite reference/glossary tables as plain declarative sentences (e.g. "X is Y").
> - Keep the original language and all technical terms, backticks, and identifiers intact.
> - Do NOT add new requirements; only restate what the table says.

The prompt is issued once per skill, with all of that skill's table blocks supplied together and the surrounding heading given as context for each. The model returns the replacement text for each block.

C TASK SUCCESS, SKILL ADHERENCE, AND EXECUTION EVIDENCE

C.1 CONFIGURATION AND DOMAIN COMPARISONS

Table 9 reports task success and skill adherence for every configuration. Task success ranges from 26.0% to 91.7%, whereas adherence ranges from 40.7% to 54.7%. Success is higher than adherence

in nine configurations; the two Grok 4.5 configurations reverse this relation. Thus, a system may be relatively successful at completing assignments while satisfying only a moderate share of skill requirements, or may follow some requirements without completing the task. Neither metric determines the other, and their numerical difference is not itself a rate of constraint violations among successful runs.

The model comparisons illustrate both forms of disagreement. Under Codex, GPT-5.6 Luna has 86.5% success and 48.8% adherence, a difference of 37.7 points, whereas Claude Opus 5 has lower success (77.1%) but higher adherence (53.0%). Under Claude Code, Qwen3.8-Max has the highest success rate, yet the 37.7-point difference between its success and its adherence remains substantial. Agreement in rank therefore does not establish agreement in what is achieved. Conversely, Grok's higher adherence than success should not be read as superior compliance: its adherence remains below that of the other configurations. The positive difference arises because its task success is lower still.

The same separation appears after aggregating by application domain. Mean success exceeds mean adherence in all ten domains, with differences from 2.0 points in Product & Project Management to 38.1 in Startup & Strategy (Figure 4(c)). These averages pool different skills and requirement compositions, so the larger gap does not by itself establish that a domain causes poorer adherence. Nor does subtracting the two means identify which successful runs violate constraints. That question requires the paired run-level comparison in the main text: 298 of the 717 accepted executions, or 41.6%, satisfy fewer than half of their assessed requirements.

Figure 3(d) is the per-configuration form of the category averages in Figure 3(c). The numeral beside each bar is that configuration's rank within its category, which Table 1 does not print. Read against the table, the numerals make the pattern reported in Section 4.2 explicit: Procedure is the weakest of the three categories in all eleven configurations, while the configurations that lead a category are not the ones that lead the next – Claude Opus 5 under Claude Code ranks first on both Procedure and Restraint but sixth on Deliverable.

Figure 6 reports the harness comparison of Figure 3(a) with one model omitted at a time.

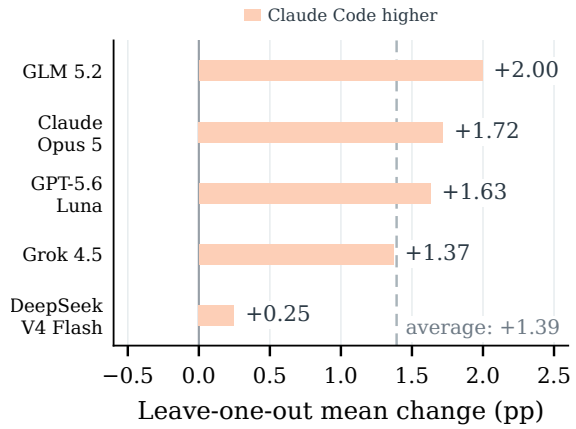


Figure 6: **Sensitivity of the harness comparison to the omitted model.** Mean difference in overall adherence between the two harnesses after excluding the named model from the five evaluated under both; the dashed line marks the mean over all five.

C.2 Skill Difficulty on Two Axes

The comparisons above are stated for configurations and for domains. Table 10 states the same comparison for individual skills, placing each of the 96 on two axes: its mean adherence over the eleven configurations, and the share of those configurations whose task rubric accepted its run. Each axis is cut at its own tertiles, so that the low band holds the skills that are harder on that axis, and the table counts how the two bandings overlap.

Table 9: **Task success and skill adherence.** Scores are reported as percentages. Δ denotes adherence minus task success, in percentage points.

Harness	Model	Success	Adher.	Δ
Codex	Claude Opus 5	77.1	53.0	-24.1
	DeepSeek V4 Flash	60.4	48.7	-11.7
	GLM 5.2	59.4	54.3	-5.1
	GPT-5.6 Luna	86.5	48.8	-37.7
	Grok 4.5	26.0	40.7	$+14.7$
Claude Code	Claude Opus 5	82.3	53.1	-29.2
	DeepSeek V4 Flash	82.3	54.7	-27.6
	GLM 5.2	59.4	53.3	-6.1
	GPT-5.6 Luna	86.5	49.3	-37.2
	Grok 4.5	35.4	42.1	$+6.7$
	Qwen3.8-Max	91.7	54.0	-37.7

Bold marks the best value within each harness in the two score columns. Δ is a difference rather than a score, so it is left unbolded. Harness names refer to the claude-code and codex CLIs, and model names to the model served in each run. "Adher." is the Overall row of Table 1, restated here so the two measures can be read off one table.

Table 10: **Skill difficulty on two axes.** Each of the 96 skills is banded by its mean adherence over the eleven configurations and by the share of those configurations whose task rubric accepted its run. Each axis is cut at its own tertiles, so the low band holds the skills that are harder on that axis. Cells give the number of skills; the last two columns give each band's means.

	Task success				Band mean	
Adherence	Low	Mid	High	Total	Adher.	Success
Low (hard)	9	10	13	32	36.3	66.8
Mid	13	10	9	32	50.5	64.8
High (easy)	7	13	12	32	63.7	72.2
All skills	29	33	34	96	50.2	67.9

Bands are adherence below 44.2%, 44.2–56.2%, and above 56.2%. Task success is accepted or rejected per configuration, so a skill's rate takes only multiples of $1/11$; the tertiles of that axis fall at at most $6/11$, 7–$8/11$, and at least $9/11$, and ties at the cuts leave the bands 29, 33, and 34 skills wide rather than 32 each. Adherence is the pass reported in Table 1, with the 19 archive-carried cells retained as elsewhere. The rows and the columns of the body each sum to 96.

The two axes carry almost no information about each other. The rank correlation between a skill's mean adherence and its success rate is $+0.05$ over the 96 skills ($p = 0.61$), and $+0.12$ with the two Grok 4.5 configurations left out ($p = 0.26$); neither is distinguishable from zero. Aggregating to the ten domain means does not concentrate the relation either ($+0.07$, $p = 0.86$). No monotone relation between a skill's adherence and its completion rate is visible at this sample size.

What the table and Figure 7 do show is that both kinds of mismatch are common. Nine skills fall in the low band on both axes and twelve in the high band on both, but twenty-two are completed in most configurations while sitting in the middle or low adherence band, and twenty are adhered to comparatively well while their task is often not completed. The widest separation is 053-go-to-market-plan, whose task is accepted in ten of the eleven configurations while 13.1% of its constraints pass; Table 11 examines one of those runs. Because the two axes are not competing measurements of one underlying quantity, a benchmark that reported either alone would call a different set of skills difficult.

C.3 A Successful Run That Satisfies Almost No Skill Constraints

The go-to-market example makes the distinction concrete (Table 11). DeepSeek V4 Flash under Claude Code produces three customer-acquisition strategies with steps, metrics, and milestones grounded in FOUNDER_CONTEXT.md. The task judge accepts all seven completion criteria, but

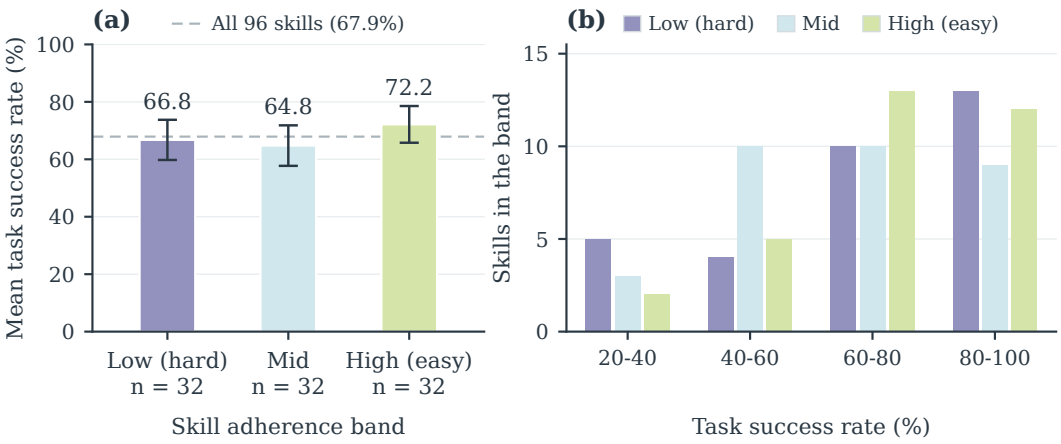


Figure 7: **Skill difficulty on two axes, band by band.** The 96 skills are split into the three adherence bands of Table 10, 32 skills each. **(a)** Each bar is the mean task success of one band over the eleven configurations. The whiskers are 95% confidence intervals on those means, using $t_{0.975,31} = 2.04$; the rule is the mean over all 96 skills, 67.9%. All three intervals cover that rule. **(b)** The same 96 skills, counted by task success in bins of twenty percentage points; the bins start at 20% because no skill's rate falls below 27.3%, and the four bars of a band sum to its 32 skills. The colours are (a)'s. Every band reaches at least 90.9%, and every band puts more than half of its skills above 60%, so the three distributions overlap across the whole scale. The spread inside a band is 64 to 73 percentage points against the 7.4 points between the band means.

only two of the skill's 16 constraints pass. The table presents two selected requirements and the corresponding evidence, rather than a complete enumeration of the rubric. This case illustrates how the evaluations differ; it is not used to estimate the frequency of any failure form.

The first requirement combines coverage and order: six dimensions of market entry must be analyzed before strategies are generated. The recorded assumptions cover three dimensions, channel fit and go-to-market motion appear only within the strategies, and market timing is absent. The assigned type records the prescribed analysis as never run: the form names the element the record lacks, and the ordering question the trace also raises is not typed separately, since the sequence cannot be read until the elements are present. The second requirement calls for Quality Checklist self-verification before presentation; the checklist is absent from both the tool log and the output. This supports a failure to establish the prescribed verification in the available record, rather than a claim about whether the model performed an unrecorded internal check. Together, the examples explain why an accepted deliverable can still fail requirements concerning its preparatory analysis and its verification.

D Domain-Level Variation

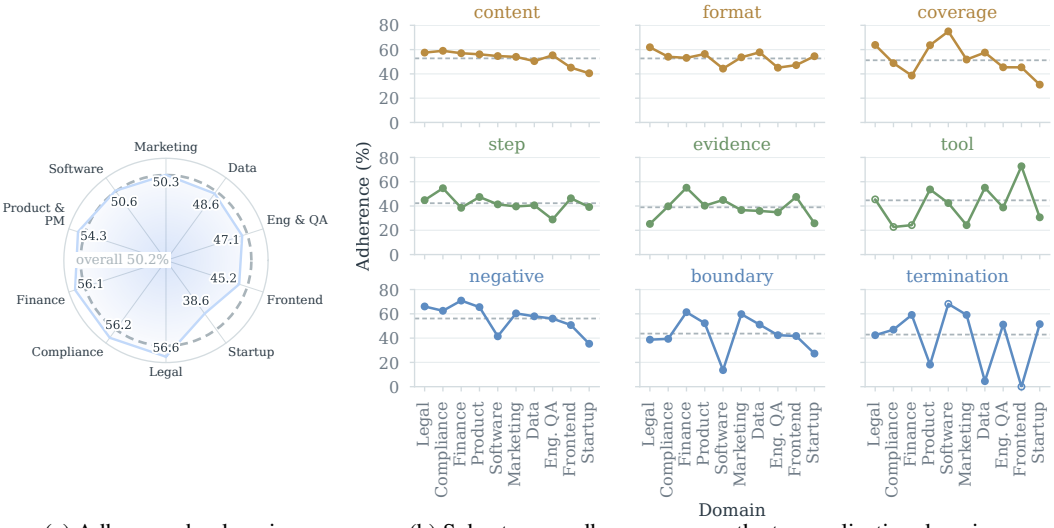
Mean adherence ranges across the ten application domains from 38.6% in Startup & Strategy to 56.6% in Legal Practice, an 18.0-point span (Figure 8(a)). Four domains fall below the overall mean of 50.2%—Startup & Strategy (38.6%), Frontend, Design & Games (45.2%), Eng Process & QA (47.1%) and Data & Analytics (48.6%). Each of these means pools 8–12 skills, and the skills within a domain differ in their requirement composition and in their individual difficulty. Membership of a domain is not a controlled condition, so we read the panel as descriptive heterogeneity rather than as an ordering of domains by difficulty; the paired run-level comparison in Section 4.3, which holds the skill fixed, is the one that isolates the gap for a single specification.

The domain means above pool requirements of nine different kinds, and Figure 8(b) splits each of them back into its subcategories. The subcategories do not move together across domains: format spans 17.6 points between its strongest and weakest domain and content 18.5, whereas termination spans 68.2 and tool 50.0. The same two subcategories, termination and tool, are carried by the fewest

Table 11: **An accepted task outcome that satisfies almost no skill constraints.** Selected requirements and evidence from skill 053-go-to-market-plan, executed by DeepSeek V4 Flash under Claude Code. The task judge accepts all seven completion criteria; the adherence evaluation rejects 14 of the 16 skill constraints. Requirements and trace descriptions are condensed; monospace strings are quoted verbatim. Failure forms follow Appendix E; where the record shows neither the action nor its artefact, the form follows what the requirement asks for, so a requirement to carry out a step is assigned the form *Step never ran*.

Task requirement	Skill requirement	Execution and artefact	Failure form
Three strategies for acquiring customers, each with steps, metrics and milestones, grounded in FOUNDER_CONTEXT.md.	Analyze market entry across six dimensions *before* generating strategies. *(step)*	Assumptions cover 3 of 6 dimensions; channel fit and GTM motion appear only inside the strategies; market timing, never.	**Step never ran**
	Complete the Quality Checklist self-verification before presenting output. *(evidence)*	Quality Checklist appears in neither the tool log nor the output.	**Step never ran**

Task success: **yes** (7 of 7 completion criteria); skill adherence **0.13**, with 2 of the 16 constraints passing.



(a) Adherence by domain

(b) Subcategory adherence across the ten application domains

Figure 8: **Domain-level variation.** One spoke per domain in (a), ordered by mean adherence, which is the value printed at each spoke and the pale blue outline, and the dashed ring is the overall adherence mean. (b) splits the same domain means into the nine requirement subcategories, one panel per subcategory, all nine drawn on a shared 0–80 axis so that panels can be read against one another, with the ten domains in one order throughout (ranked by overall adherence, as in (a)); the dashed rule there is the subcategory's mean over all 96 skills. In both panels a domain mean averages that domain's 8–12 skills over the eleven configurations, counting only the skills that carry a judged requirement in the subcategory, so the five models under both harnesses count twice; hollow points in (b) mark the cells that average one or two skills. Domains differ in size and in requirement mixture, so both panels describe heterogeneity rather than ranking domains by difficulty.

skills (34 and 33 of 96), so the cells that average only one or two skills are drawn hollow rather than printed as though they described a domain.

E FAILURE FORMS IN SUCCESSFUL BUT LOW-ADHERENCE RUNS

E.1 ANALYSIS POPULATION AND MODE ASSIGNMENT

The failure analysis is restricted to 298 runs that pass the task-success evaluation but satisfy fewer than half of their assessed skill constraints. These runs fail 1,471 of the 2,194 constraints, and 4,347 failed checks at the (run, constraint) unit that the main text uses. Table 12 gives the seven forms with the nine requirement types weighted equally, Table 13 counts the same checks under the observed weighting, and Table 14 counts the requirements.

Every failed check is assigned exactly one form, by a forced-choice pass over the finding of an earlier pass that read the run's evidence. A subcategory identifies what the requirement governs and a form describes what was assigned, so the two cross. The same form appears under several subcategories, and no subcategory is characterized by a single form. Figure 4(d) gives this assignment by subcategory in the main text; Figure 9 and Tables 12, 13, and 14 give it in full. The failure unit repeats a skill once per configuration it was run in, so a requirement of a widely run skill weighs more than one of a skill run once; the requirement count removes that weighting, and the equal weighting of Table 12 removes the subcategory's own size, each at the cost described below.

Figure 9 gives the same typing as Figure 4(d) at a finer resolution, one cell per subcategory and form, and prints the exact percentages behind that figure's bars. The nine subcategory rows are the same in both, but the pooled row here is weighted by the subcategories' check counts, so it is the summary of Table 13 rather than the equal-weighted Table 12. Each subcategory row is normalized by its own number of failed checks, whereas the pooled row is weighted by those counts. Among the subcategories, content contributes 1,927 failed checks and termination 87, so the same within-subcategory percentage would correspond to very different numbers of checks. The shared color scale supports comparison of proportions across rows; it does not remove this difference in sample size. These percentages describe the composition of failures in the selected subset, not the probability that a requirement fails across all benchmark executions; Table 15 gives that probability within the same subset, and the two need not order the subcategories alike.

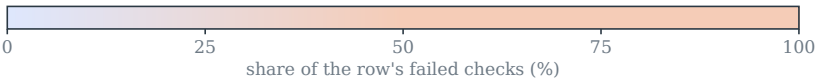
E.2 THE FORM ASSIGNED TO EACH FAILED CHECK

Grouping the seven assigned forms by whether the requirement was met at all separates two cases that a single composition merges, and it is the one comparison on this axis whose leading direction the weighting decides. Under the observed weighting the larger group is the requirements that were produced without satisfying the check, 52.9% against 47.1% never produced; giving the nine requirement types one vote each reverses it, to 44.7% against 55.3%, and a cluster bootstrap over the 77 skills puts the two intervals at $[51.3, 59.4]$ and $[40.6, 48.7]$, which do not overlap. The reversal is a property of the pooled comparison rather than of every configuration: counted per check, requirements never produced are the larger group in four of the eleven and requirements produced without satisfying the check in the other seven, so within a single configuration neither group's lead is fixed. It is also not an artifact of pooling the subcategories. Within the subcategories the direction is set by what each one can observe: content and format lean the same way as the pooled count (42.4% and 34.1% never produced against 57.6% and 65.9% produced), while coverage leans further still, 70.7% of its failures present but incomplete, mislabelled, or in the wrong place against 29.3% never produced. Requirements about actions behave the other way, and more sharply: 68.4% of failed step checks and 96.1% of failed tool checks are work the execution record never shows, and negative constraints, which forbid an action rather than require one, lean the same way at 59.5%. Under the observed weighting the count follows the mix of requirement types a skill happens to state, and most requirements are content or format; under equal weighting the small action-bound subcategories, which almost never fail as anything else, count as much as content does.

Counting the same failures once per requirement instead of once per failed check gives the smaller table below. That reduction is not mechanical, because 1,021 of the 1,471 constraints failed in more than one run and 609 were assigned differently by at least one of them. Each is given the form assigned by the majority of its failed checks, with exact ties broken by the order in which the seven forms are listed; 229 constraints reach such a tie, and the rule leaves 943 of the 4,347 checks unrepresented in the table below. The group totals move with that choice: as printed, 54.9% of constraints are never produced against 45.1% produced but not as required, but under the opposite

Figure 9: **The form assigned to each failed check, by subcategory.** Entries are row-wise percentages among the 4,347 failed checks of 298 successful runs with adherence below 50%, each counted once per run that failed it. Each failed check is assigned one form, by a forced-choice pass over the finding of an earlier evidence pass. n gives the number of failed checks in each subcategory; the pooled row weights subcategories by these counts, so it is Table 13 and not the equal-weighted Table 12. A shared 0–100 color scale permits comparison of proportions across rows; bold marks each row maximum. Percentages may differ slightly from 100 after rounding.

	Content missing	Wrong wording	Wrong location	Partly done	Step never ran	Wrong structure	File missing	*n*
content	**37.8**	26.1	13.5	17.4	1.5	0.6	3.1	1927
format	**27.3**	25.6	20.3	15.1	0.5	4.9	6.3	649
step	15.0	11.9	6.8	11.3	**51.5**	1.6	1.8	487
coverage	23.9	25.0	7.8	**37.9**	4.0	0.0	1.4	348
negative	**41.8**	17.0	9.0	13.2	15.8	1.3	1.9	311
evidence	27.5	21.1	10.1	11.7	**27.9**	0.8	0.8	247
boundary	**47.3**	25.0	11.2	10.6	5.3	0.0	0.5	188
tool	6.8	2.9	1.0	0.0	**89.3**	0.0	0.0	103
termination	**41.4**	24.1	3.4	13.8	16.1	0.0	1.1	87
All	**32.0**	22.8	12.2	16.6	12.2	1.3	2.9	4347



Failure forms: **Content missing** the required element appears in neither the final answer nor any file the run wrote; **Wrong wording** it appears, but not in the literal form the check searched for; **Wrong location** it appears, but in a file or place other than the one searched; **Partly done** some of the items the requirement names are present and others are not; **Step never ran** the execution record shows no evidence of the prescribed step; **Wrong structure** the element is present, but its format or structure does not match; **File missing** a required file was never created. Assignment is forced: each failed check in this sample takes exactly one of the seven forms, so the set is complete over the failures typed here. It describes the forms observed in this sample and is not a claim to cover every way a specification can be contravened.

tie-break they reverse, to 47.2% against 52.8%, and allocating each constraint fractionally across the forms it took instead of giving it to the majority gives 50.4% against 49.6%. Neither choice dominates the other: on the group totals the three requirement-level rules span 47.2% to 54.9% and the two weightings of the check-level table span 47.1% to 55.3%, so the unit and the weighting move this comparison by comparable amounts, and each has to be stated rather than assumed. The paper reports the check-level table under both weightings because each of its two entries follows from a rule that is fixed before the numbers are seen—assign every check its form and count—where the requirement-level table adds a third convention that the data do not supply. It is reported here because it shows how much the unit is worth on its own.

Rebasing those counts on the 6,687 checks, as the *per check* column of Table 13 does, leaves the ranking of the seven unchanged: every form is divided by the same number. What the rate does show is that the largest form, *Content missing*, occurs on 20.8% of checks rather than the 32.0% its share of the failures suggests. The composition is still the more useful of the two for this axis, because a form's count is inflated not by how often its requirements are assessed but by which subcategories produce it: five of the seven forms draw between 46% and 52% of their counts from content, against content's own 44.3% share of the failures, so for those five the observed weighting largely restates content's form mix. The two that do not are the ones bound to a particular kind of requirement, *Step*

Table 12: **The form assigned to each failed check, with the nine requirement types weighted equally.** Each failed check of the selected runs is assigned one form from the scheme; a form's share is the unweighted mean of the nine subcategories' own compositions—for form f in subcategory c, with n_{cf} its failures there and $n_c = \sum_f n_{cf}$, the share is $P(f) = \frac{1}{9}\sum_{c=1}^{9} n_{cf}/n_c$—not its share of the failures that occurred. The group headings collect the seven forms by whether the requirement was met at all; the italic rows are that group's subdivision. Table 13 gives the same seven under the observed weighting, which counts each failed check once.

Form assigned	Share
Required content, actions, or files absent	55.3%
Content missing	29.9%
Step never ran	23.5%
File missing	1.9%
Produced but not as required	44.7%
Wrong wording	19.9%
Partly done	14.6%
Wrong location	9.2%
Wrong structure	1.0%
Total	**100.0%**

Percentages are the unweighted mean of the nine subcategories' own compositions, so a form's share is what its profile looks like when each requirement type counts once, not its share of the failures that occurred. Each row sums to 100% over the seven forms and the italic rows of a group add to the printed group, though an individual value may differ from its exact share after rounding. The pool behind the means is the 4,347 failed checks of the 298 selected runs, counted once per (run, constraint) pair; each check is assigned exactly one form. *Content missing* is assigned only when the required element appears in neither the final answer nor any file the run wrote.

Table 13: **The form assigned to each failed check, counted once per failed check.** The 4,347 failed checks of the 298 selected runs of Table 12, each counted once for every run that failed it, so a requirement that several runs failed weighs as many times as it failed. *Per check* rebases the same counts on the 6,687 checks these runs produced, so a form's rate is the share of all checks that ended in it; the seven rates and the 35.0% of checks that passed sum to 100%. Table 12 reports the same seven forms under the opposite weighting, which gives the nine requirement types one vote each.

Form assigned	Checks	Share	Per check
Required content, actions, or files absent	2,047	47.1%	30.6%
Content missing	1,391	32.0%	20.8%
Step never ran	531	12.2%	7.9%
File missing	125	2.9%	1.9%
Produced but not as required	2,300	52.9%	34.4%
Wrong wording	990	22.8%	14.8%
Partly done	722	16.6%	10.8%
Wrong location	530	12.2%	7.9%
Wrong structure	58	1.3%	0.9%
Total	**4,347**	**100.0%**	**65.0%**

never ran drawing 5% of its counts from content and *Wrong structure* 21%. That is what the equal weighting in Table 12 removes, by reporting each requirement type's own composition in place of the pooled one.

Those counts are shaped by how many requirements each subcategory contributes, so they have to be read against the pool. Deliverable requirements are 67.4% of the 2,194 assessed constraints and 67.3% of the failed checks. A form's total therefore reflects mostly how many requirements of that kind a skill states. *Content missing* leads under every weighting tried here, so its position is not an artifact of that alone, but the weighting sets how far ahead it is: 32.0% against 22.8% counted per check, 40.8% against 21.3% per requirement, and 29.9% against 23.5% with the nine requirement types weighted equally. A subcategory's count is not on its own a measure of how readily its requirements fail, and the two are not interchangeable: Table 15 gives the rate per check, on which step leads at 1.16 times the pooled rate and negative trails at 0.84, while tool sits below the pooled rate despite taking more than its share of the counts.

Table 14: **The form assigned to each failed check, counted once per requirement.** The 1,471 constraints that failed in at least one selected run, by the form assigned to the checks that failed. Each constraint is given the form assigned by the majority of its failed checks. This is Table 13 counted once per requirement rather than once per failed check, so a requirement that several runs failed counts once.

Form assigned	Constraints	Share
Required content, actions, or files absent	807	54.9%
Content missing	600	40.8%
Step never ran	165	11.2%
File missing	42	2.9%
Produced but not as required	664	45.1%
Wrong wording	314	21.3%
Partly done	193	13.1%
Wrong location	144	9.8%
Wrong structure	13	0.9%
Total	**1,471**	**100.0%**

The grouping neither assigns severity nor establishes a single cause such as forgetting the instruction. It is particularly relevant for format, where substantive content may be present throughout despite a failed exact-string requirement. *Wrong wording* is the form this produces and the second-largest under the observed weighting: 990 checks, 22.8% of the total, of which 503 fall under content and 166 under format; equal weighting puts it third, at 19.9%, behind *Content missing* and *Step never ran*. It is assigned when the required element is present but the check searched for a different string—a hyphen, a synonym, a different phrasing, a differently spelled heading, or another language. An exact-form check is used only where the specification itself names the element it wants, whether a label, a format, or a statement such as a stop rule, and the specification's own wording for that element is among the strings it searches for; the form records a substitution of the prescribed form, not a paraphrase of a requirement left open.

A count is not a rate. Table 15 divides each subcategory's failures by its checks, so that every check counts as one occasion on which such a requirement could have been satisfied: a subcategory is assessed once per requirement *per run of a skill that states it*, and the nine are not assessed equally often.

Table 15: **Failure rate per check, by subcategory.** The 4,347 failures of Table 13 divided by the 6,687 checks the same 298 runs produced. A check is one (run, constraint) pair, so a requirement is assessed once per run of a skill that states it; *checks per req.* is that count against the 2,194 constraints of Table 8, and the nine differ because their requirements sit in skills that appear in these runs different numbers of times. The pooled rate is 65.0%; the families aggregate the rows to Deliverable 65.2%, Procedure 69.7%, and Restraint 58.4%.

Subcategory	Constraints	Checks	Checks per req.	Failure rate	vs pooled
content	1,012	3,028	2.99	63.6%	0.98
format	314	929	2.96	69.9%	1.07
coverage	152	526	3.46	66.2%	1.02
step	217	648	2.99	75.2%	1.16
evidence	126	382	3.03	64.7%	0.99
tool	47	171	3.64	60.2%	0.93
negative	188	572	3.04	54.4%	0.84
boundary	100	299	2.99	62.9%	0.97
termination	38	132	3.47	65.9%	1.01
Total	**2,194**	**6,687**	**3.05**	**65.0%**	**1.00**

The rates are pooled over skills, so a subcategory whose requirements sit in harder skills is charged for that difficulty; within skills, the ordering is resolved for no subcategory except negative, whose requirements fail 11.1 points below their skill's other requirements (95% interval -19.3 to -3.0, clustered by skill). They are also conditional on the selected runs, so they describe requirements already being failed in successful executions rather than a property of the nine types across all runs.

E.3 Actions, Evidence, and the Deliverable Forms

Requirements that name an action fail in a way that leaves little room for interpretation. Of failed step checks, 51.5% are steps the execution record never shows, and of failed tool checks, 89.3% are. The evidence subcategory divides between steps never run (27.9%) and content missing (27.5%), so a requirement to ground a claim in a source is failed either by a check that was never carried out or by a result reported without its support. That 0.4-point lead, and the 1.7-point lead content missing holds over wrong wording among format checks, are the two narrowest in the table; elsewhere the leading form is ahead by at least 5 points. These are point estimates, and a cluster bootstrap over the 77 skills resolves a leading form above its nearest rival in step, tool, negative, and boundary, but not in content, format, coverage, evidence, or termination, where the top two are not separated at this sample size; the two narrow leads above are the extreme case of that. Because deliverables carry the larger share of the constraint set, these procedural failures remain a small part of the pooled composition even where they dominate their own subcategory: an accepted final answer can pass its outcome rubric while the recorded support and the checks the skill requires are absent.

The Restraint subcategories are the ones the record is least able to settle. Negative constraints are assigned the deliverable forms, content missing leading at 41.8%. Boundary and termination fall on the output forms as well: content missing leads both, at 47.3% and 41.4%, with wrong wording the runner-up in each. A scope requirement is therefore more often failed by the absence of something the check looked for than by a distinguishable overreach.

That the record cannot settle whether a restriction would have been respected does not make these a weaker kind of failure. What the record settles varies by requirement: a prohibition can be failed by an action the record shows, while a stopping rule the deliverable must state is failed by the absence of that statement, and it is the second case that reaches the output forms. The A/B-testing skill requires the plan to halt the test if a guardrail metric turns significantly negative. Six runs failed that requirement. Each of them wrote guardrail metrics, and each wrote a rule triggered by them—keep the control, do not ship—but every one of those rules governs the shipping decision, and none of them stops the test while it is running. Those runs wrote a rule the skill did not ask for and omitted the one it did. The check reads the plan and finds no stop condition, so the form assigned is Content missing, because the element the requirement names is absent from what the run produced. The same omission appears as Wrong wording when a run states the halt in other words than the check searches for, and as Wrong location when it states it somewhere the check does not read. The requirement text is why: of the 130 negative checks assigned Content missing, 54% state a positive obligation alongside their prohibition, and 46% of the 89 boundary checks do, against 8.8% of content checks. The subcategory records which requirement was missed; the form records what the record shows in its place.

Because a form's concentration is tied to what its subcategory can observe and to the evidence its checks draw on, the composition reflects both the executions and the checks that observe them. The composition is therefore read here for what it identifies directly: which observable failures remain once task success is established.

E.4 One Worked Example of Each Form

The seven forms are separated by what the run did rather than by how much of the task it completed, so the shortest way to fix their meanings is an example of each. Table 16 gives one per form. Each row is a requirement the skill states explicitly and a run that did not satisfy it, and each is reported by the check directly rather than inferred from the record.

Three properties of these examples are worth stating, because they are what the counts in Table 13 sum over. The unit is one requirement, not one deliverable: the go-to-market plan of Appendix C is an accepted answer that fails 14 of its 16 requirements, and the WCAG audit above produces a usable report while failing the header field. The requirement is stated by the skill and not inferred from the task, so a run can satisfy every completion criterion and still fail the row. And the assignment records what the check found, so the run that searches for *Quick Wins* and meets *CLASS A* is failing the requirement as written even where a reader would call the two labels equivalent.

Requirements that name a label, a heading, or a file are the ones a run most often satisfies in substance and fails in form. A tier that fixes the same problems on the same day is doing what *Quick*

Table 16: **One example of each assigned form.** Each row quotes a requirement stated explicitly by the skill named and what the run did instead. Requirements and trace descriptions are condensed; monospace strings are quoted verbatim. The examples come from different skills and configurations, and the constraint identifier is that of the adherence rubric.

Form	What the skill required	What the run did
Content missing	*281-wcag-accessibility-audit, C44:* the report header includes a `WCAG Version` field.	The header carries no version field; the string appears nowhere in the output.
Wrong wording	*148-signup-flow-cro, F02:* the same-day fixes form a tier labelled *Quick Wins*.	The tier exists, but is labelled *CLASS A — Fixable immediately*; the required name appears nowhere.
Wrong location	*116-tpl-situacao-sprint-planning-tecnico, C06:* the plan contains a `### Sprint Goal` section.	The goal is stated in the chat reply only; the plan body has no such section.
Partly done	*010-server-management, C15:* monitoring covers CPU, memory, disk and network.	Only disk appears, and only in a log-rotation context; the other three are absent from the monitoring layer.
Step never ran	*042-forecast-premortem, C03:* read the bundled method document under `resources/` before writing the failure narrative.	The tool log shows reads of two other bundled resources and none of this one.
Wrong structure	*015-outreach-specialist, C08:* the output opens with the `## Outreach Sequence` heading.	The heading is present, but process narration precedes it: *I'll review the outreach workflow . . . before drafting*.
File missing	*062-plugin-settings, C01:* create the local settings file, at the project root under `file/setting`.	The file is never created.

Wins was asked to do, and the requirement is still unmet; the same holds for a heading written one level deep, for a section written in the reply rather than in the deliverable, and for a file shipped as a template for the user to copy rather than created. These are not incidental to the measure: they are the reason a run that passes every completion criterion can still be missing part of its specification. Table 17 follows two of these requirements end to end, from the sentence the skill states to the verdict the check records, and shows the two links the examples above leave implicit: where in the skill the requirement comes from, and what the check searches for.

Table 17: **The chain from a skill's sentence to a recorded failure.** Each column follows one requirement: the sentence the skill states, the constraint the adherence rubric extracts from it, what the check searches for, and what the run produced instead. Skill and constraint wording is condensed; quoted strings are those the check searches for.

	148-signup-flow-cro, F02	**158-ab-test-setup, C27**
The skill states	Recommendations are always organized as *Quick Wins* → *High Impact* → *Test Hypotheses* — never a flat list. *(SKILL.md, Communication)*	Guardrail metrics are the things that should not get worse. Stop the test if significantly negative. *(SKILL.md, Guardrail Metrics)*
The rubric extracts	Recommended changes include a Quick Wins tier labeled as same-day fixes. *(format)*	The plan includes a stop condition: halt the test if a guardrail metric is significantly negative. *(termination)*
The check reads	The output, or any file the run wrote, must contain `Quick Wins`; a second leg puts the same requirement to a judge, which reads the recommendations section.	The plan must contain `stop the test`, `halt the test`, `serious guardrail deteriorates`, or `unacceptable harm`; a second leg puts the same requirement to a judge, which reads the plan's guardrail guidance.
The run produced	A same-day tier headed *CLASS A — Fixable immediately*. The tier is there; the string `Quick Wins` appears nowhere.	Guardrail metrics listed, and a rule keyed to them — keep the control, do not ship — but the rule governs the shipping decision and never halts the test while it runs.

Table 18: **Selected specification-property associations.** Differences in criterion pass rate, in percentage points, with 95% confidence intervals. These are observational estimates, not gains established by rewriting a skill; the final row expresses the linear obligation-count estimate per one fewer obligation. Estimates should not be added.

Specification property	Δ pp [95% CI]
Requirement appears in the first 800 words of SKILL.md.	$+5.1$ $[+2.0, +8.3]$
Required output string appears three to four times in SKILL.md.	$+10.2$ $[+3.5, +17.1]$
Required format string is quoted verbatim rather than paraphrased.	$+9.9$ $[+0.9, +18.8]$
One fewer obligation in the criterion (linear estimate).	$+2.9$ $[+1.5, +4.2]$

F Specification Properties and Targeted Edits

F.1 Comparison Units and Interpretation

This analysis asks whether the expression of a requirement is associated with its satisfaction, and whether editing that expression improves adherence. The observational comparisons are made within skills while accounting for model differences, with uncertainty clustered by skill. This reduces confounding by differences in overall skill difficulty, but does not make requirement properties randomly assigned. Authors may place or repeat requirements according to their importance or complexity, and those characteristics may also affect satisfaction. The resulting pass-rate differences are associations between individual requirements, not expected gains from applying an edit to a skill.

The analysis sample and aggregation differ from the main benchmark scores. Table 19 covers the same 2,194 criteria as the full taxonomy. Its raw pass rates weight criterion–model cells, while the main scores first average within skills. Consequently, raw pass rates in the property analysis need not reproduce the macro-averaged scores in the main text. Throughout this appendix *native* denotes the six configurations run under the Claude Code CLI (Table 1); the five Codex configurations are a separate comparison, and the deterministic script checks a third channel.

F.2 Placement, Repetition, and Requirement Complexity

Position-dependent performance in long-context tasks (Liu et al., 2024a) motivates examining where a requirement appears, although that finding does not establish the same mechanism in skill execution. Requirements in the first 800 words of SKILL.md have an estimated pass rate 5.1 points higher than other requirements within the same skill (95% CI $[2.0, 8.3]$). The estimate is 11.7 points for step, evidence, boundary, and termination requirements. Required literal output strings appearing three or four times have a 10.2-point higher pass rate ($[3.5, 17.1]$), rising to 15.0 points for strings of two or three words. These comparisons are consistent with a relationship between how a requirement is presented and whether it is followed, but do not identify the mechanism. Repetition is also not monotonically beneficial: other repetition bands have smaller estimates.

Table 18 summarizes selected associations. Quoting a required format string verbatim rather than paraphrasing it is associated with a 9.9-point higher pass rate ($[0.9, 18.8]$). Additional obligations within a criterion are associated with lower pass rates, at approximately 2.9 points per obligation. The latter is a linear estimate from rubric wording, not a direct comparison between all single-obligation and all multi-obligation requirements. Its interpretation is also sensitive to the evaluation channel: the corresponding script estimate reverses sign. The direction of this effect therefore depends on what the evaluation holds fixed, and separating a criterion from its wording changes both at once.

F.3 Subcategory Differences within Skills

Comparisons within skills retain several of the differences seen in the aggregate subcategory scores. The evidence subcategory is 7.9 points below the skill's other requirements ($[-13.1, -2.8]$), boundary 7.5 points below ($[-14.2, -0.9]$), and step 6.9 points below ($[-11.7, -2.2]$). The negative subcategory is 4.8 points above ($[0.1, 9.3]$). These estimates indicate that the procedural and scope-related shortfalls are not explained solely by those subcategories appearing in harder skills. They do

Table 19: **Subcategory differences within skills.** Subcategory labels follow Table 8; n counts the criteria represented in this analysis. Pass rate is the raw share of satisfied criterion–model cells, whereas Δ compares a subcategory with other requirements within the same skill. Intervals are clustered by skill. These cell-level rates use a different aggregation from the macro-averaged main results.

criterion subcategory	n	pass rate	Δ pp	95% CI
negative	188	54%	$+4.76$	$[+0.1, +9.3]$
content	1,012	53%	$+2.75$	$[+1.2, +4.3]$
format	314	51%	$+0.81$	$[-2.5, +4.2]$
tool	47	44%	-1.16	$[-9.4, +7.7]$
coverage	152	45%	-2.41	$[-7.9, +2.8]$
step	217	43%	-6.88	$[-11.7, -2.2]$
termination	38	40%	-7.25	$[-18.7, +3.8]$
boundary	100	44%	-7.48	$[-14.2, -0.9]$
evidence	126	41%	-7.92	$[-13.1, -2.8]$

The analysis covers all 2,194 criteria of the full taxonomy, judged across the six native pools. A few criteria were not judged in every pool, so the criterion–model panel is not rectangular; the counts above describe the property-analysis sample and should not replace the benchmark totals. Seven of nine subcategory estimates retain their sign on script checks and eight on Codex; agreement across these comparisons supports the descriptive pattern but does not rule out shared measurement limitations.

not establish that the subcategory itself causes the difference: individual requirements can still differ in complexity, observability, or how they are judged.

Uncertainty is substantial for several subcategories. The termination subcategory has a point estimate of -7.25 points, but its interval spans $[-18.7, 3.8]$; tool, coverage, and format also have intervals crossing zero. Thus, the within-skill analysis supports some broad differences without determining a precise ordering of all nine subcategories. Agreement in sign across script checks and the Codex comparison provides additional descriptive support, and the evidence, boundary, and step differences keep that sign on both, while differences between channels caution against attributing the full pattern to agent behavior alone.

F.4 Targeted Placement and Repetition Edits

To test whether the observed presentation effects translate into better adherence, 38 skills are rerun after targeted placement and repetition edits under a single model and harness. These are the skills whose existing text admitted such an edit; where the placement or repetition property was already satisfied, or no candidate edit applied, the skill is left out of the intervention. The model, judge, and evaluation criteria are held unchanged; the intervention alters the skill text rather than the requirements used to score it. Keeping the rubric fixed is essential here: otherwise, an apparent improvement could reflect a change in what counts as success rather than improved compliance with the same obligations.

Mean adherence rises from 50.3% to 52.0%, an estimated gain of 1.66 percentage points with a 95% interval of $[-3.1, 6.5]$, paired by skill against the same skills before the edits. Eighteen of the 38 skills improve, seventeen decline, and three are unchanged, so the positive mean sits on balanced directions rather than on a consistent shift. Requirements the edits did not touch, judged in the same executions, number 795 and move by less than half a point, so the two runs do not differ by a general shift in judged adherence. With one execution per condition, execution-to-execution variation is not separately estimated.

The two analyses answer different questions, and the difference bounds the property rather than contradicting it. The comparisons above describe specifications as written and find placement and repetition associated with higher satisfaction across them. The edit experiment holds the rubric fixed and asks whether a specification acquires that association by rewriting its own text; with small estimates and balanced directions, the association reads as a description of how these specifications are composed rather than as a lever that a rewrite pulls. Appendix G carries the wider set of compar-

Table 20: **Specification properties and criterion pass rate.** Reported within-skill estimates from the specification-property analysis. Δ is a pass-rate difference in percentage points; intervals are clustered by skill. The last three columns report the native evaluation pool, the Codex comparison, and deterministic script checks. Blocks group placement and repetition, requirement wording, less conclusive associations, and document-level edit comparisons. These estimates do not establish causal effects.

property	Δ pp	95% CI or p	native	codex	script
Requirements in the first 800 words	$+5.11$	$[+2.0, +8.3]$	$+5.1$	$+5.8$	$+3.9$
among step, evidence, boundary, termination	$+11.66$	$[+3.6, +19.5]$	—	—	—
Literal output string written 3-4 times	$+10.16$	$[+3.5, +17.1]$	$+10.2$	$+11.5$	$+15.6$
for strings of two to three words	$+15.04$	$[+6.1, +24.5]$	$+15.0$	$+13.1$	$+17.7$
Format string copied verbatim, not paraphrased	$+9.90$	$[+0.9, +18.8]$	$+9.9$	$+5.1$	$+1.2$
for each / *one per* in a requirement	-15.41	$[-24.0, -6.6]$	-15.4	-11.1	-0.7
all / *every* in a requirement	-6.93	$[-11.5, -2.3]$	-6.9	-5.8	$+1.7$
then / *after* / *next* in a step line	-6.55	—	-6.6	-7.6	-18.1
Each additional obligation in one requirement	-2.87	$[-4.20, -1.54]$	-2.9	-2.9	$+1.4$
Tool or command named in a step line	$+6.93$	$p = 0.083$	$+6.9$	$+17.7$	$+0.5$
Numbered rather than unordered lists	$+8.74$	$p = 0.140$	$+8.7$	$+5.7$	$+14.3$
Requirements in the final 150 words	-6.05	$[-11.8, +0.4]$	-6.1	$+1.1$	-5.8
Verification or checklist section added	$+0.05$	$[-12.3, +13.3]$	$+0.1$	-4.9	-7.4
Constraints or rules section added	-1.15	$[-9.3, +7.2]$	—	—	—
Examples section added	$+2.10$	$[-8.2, +12.7]$	—	-12.6	—
More code blocks	-0.35	$[-9.0, +9.3]$	—	—	—
More conditional branches	-1.18	$[-16.2, +13.8]$	—	—	—

Native, Codex, and script estimates assess the same property under different evaluation settings or channels; their agreement is a robustness check, not proof that measurement effects are absent. Dashes indicate unreported estimates. The sequencing estimate recomputed from rubric text is -6.20 points (95% CI $[-9.8, -2.8]$; permutation $p < 0.001$). The final block reports difference-in-differences comparisons between targeted and other subcategories. Property estimates are not additive. Two rows carry a permutation p-value in place of an interval. Appendix G discusses channel differences, negative controls, and alternative explanations.

isons and checks, and broader, evaluation-guided skill refinement (Gautam et al., 2026) is a separate question from the targeted edits tested here.

G Additional Specification Analyses and Robustness Checks

G.1 Full Property Comparisons and Evaluation Channels

Table 20 places the selected findings alongside other candidate properties. The native estimates are accompanied by the Codex comparison and by deterministic script checks, which assess mechanically verifiable aspects without consulting the LLM judge. These comparisons help identify findings that depend on one setting or assessment method. They do not hold every factor fixed: the channels can differ in coverage, sensitivity, and which aspects of a requirement they measure. Agreement therefore strengthens the descriptive evidence without eliminating shared confounding, while disagreement does not by itself establish whether the agent or the evaluator is responsible.

Placement and repetition are positive in all three reported comparisons. Early placement is associated with $+5.1$, $+5.8$, and $+3.9$ points in the native, Codex, and script columns; three-to-four-fold literal repetition with $+10.2$, $+11.5$, and $+15.6$. Other properties are more channel-dependent. Verbatim format strings have a $+9.9$-point native estimate but only $+1.2$ on scripts, while the negative association with additional obligations reverses to $+1.4$ on scripts. These differences limit claims about a general effect shared across all forms of assessment.

G.2 Quantifiers, Sequencing, and Compound Requirements

Open-quantified requirements using *for each* or *one per* have a -15.41-point association with pass rate ($[-24.0, -6.6]$; permutation $p < 0.001$). Adding subcategory indicators leaves an estimate of -16.6 points, so the association is not absorbed by subcategory composition alone. On script checks, however, it is only -0.7 points. One possible explanation is that an existence check detects an action without establishing that it covers every required item; another is that the judged and scripted subsets measure different aspects of compliance. The comparison cannot distinguish these explanations. Requirements using *all* or *every* likewise have a negative native estimate (-6.93 points), but a positive script estimate ($+1.7$).

Sequencing terms—*then*, *after*, and *next*—are associated with lower pass rates in all three reported comparisons: -6.6 points natively, -7.6 on Codex, and -18.1 on scripts. Recomputing the feature from rubric text gives -6.20 points ($[-9.8, -2.8]$; permutation $p < 0.001$). The negative sign on scripts makes an explanation specific to the LLM judge less sufficient, but does not rule out measurement effects or differences in the demands of sequential requirements. This result concerns the requirements expressed with sequencing terms, not the causal effect of adding those words.

The obligation-count analysis provides a related comparison. A single-obligation criterion is met 1.7 points more often than its skill's average, while the estimates decrease across five bins from $+1.73$ for one obligation to -23.13 for six or seven. The linear estimate is -2.87 points per additional obligation ($[-4.20, -1.54]$; permutation $p < 0.001$), and remains -2.24 after accounting for requirement length. Codex reproduces the downward pattern, and seven of eight examined subcategories have negative within-subcategory slopes. Nevertheless, the script estimate reverses sign, and the count is taken from rubric text rather than the carrying instruction in SKILL.md. The data therefore do not establish whether the additional obligations originate in the skill or its decomposition into evaluation criteria. Nor does obligation count explain the subcategory ranking: its aggregate relationship with that ranking points in the opposite direction. Treating this finding as a recommendation to split criteria would conflate a change to the evaluator with a change to agent execution.

G.3 Negative Controls

The properties above are candidates, and a candidate can correlate with judged success without naming anything an author could act on. Three controls test that, each closing one way a spurious property could pass. Matching the tokenization used to identify requirements and features cuts the apparent benefit of lists over prose from $+4.36$ to $+0.79$. A sham feature built from another skill's references reproduces 93% of the apparent association with bundled reference files. Restricting to names the rubric actually checks leaves naming files in SKILL.md at $+0.012$ ($p = 0.907$). What the controls protect is the positive estimates above: a property that holds under all three is not the same kind of result as one that does not.

Replacing *should* with *must* produces estimates with opposite signs between easier and harder requirements; the reported contrast is $+15.22$ points ($[3.9, 26.0]$). This heterogeneity does not support a uniform recommendation to strengthen modal language. Naming a tool or command in a step has a positive point estimate of $+6.93$ points but $p = 0.083$, and numbered lists have $+8.74$ points with $p = 0.140$. Requirements in the final 150 words have an estimate of -6.05 points with an interval crossing zero ($[-11.8, 0.4]$) and a positive Codex estimate. Each of these remains open rather than settled in either direction.

Document-level edits also provide no clear average gain in the reported comparisons. The difference-in-differences estimates contrast subcategories targeted by an edit with other subcategories: adding a verification section yields $+0.05$ points ($[-12.3, 13.3]$), a constraints section -1.15 ($[-9.3, 7.2]$), and an examples section $+2.10$ ($[-8.2, 12.7]$). More code blocks and conditional branches likewise have wide intervals crossing zero. Bundling additional reference material or scripts gives -0.30 points ($[-8.8, 8.1]$). These intervals leave considerable uncertainty; they do not establish equivalence or show that such resources are ineffective in every setting. One intentionally mismatched subcategory comparison produces $+9.24$ points ($p = 0.033$), the only nominally significant result among seventeen negative-control comparisons, close to the rate expected by chance at the 0.05 level. With many candidate properties examined, isolated nominal significance warrants caution.

Table 21: **Comparison with representative benchmarks.** ✓ indicates an explicitly included feature in the reported evaluation protocol. Hierarchical constraint scoring denotes adherence scores at both coarse and fine levels of a nested requirement taxonomy; SkillIF reports scores across three categories and nine constraint subcategories.

Benchmark	Reusable skill packages	Environment execution	Explicit adherence scoring	Hierarchical constraint scoring
IFEval	×	×	✓	×
AgentIF	×	×	✓	×
AgentIF-OneDay	×	✓	✓	×
SkillsBench	✓	✓	×	×
SkillIF (ours)	✓	✓	✓	✓

G.4 Aggregation and Interpretation

Subcategory composition affects pooled property estimates. The content subcategory accounts for approximately 46% of the full constraint set, so a property relevant to a smaller subcategory can have a weak pooled association even when its subcategory-specific estimate is larger. For example, the reported comparison for including a requirement in SKILL.md is $+0.44$ points when pooled ($p = 0.78$), but $+9.90$ when restricted to format ($p = 0.003$). The property estimates also overlap and tend to be smaller where other properties are prominent; adding them would not estimate the effect of a combined rewrite.

Across these analyses, the estimates that hold in every channel this appendix reports are the ones the property analysis rests on: how a requirement is placed, repeated, and worded is associated with whether it is satisfied, and placement and repetition carry the same sign under the native evaluation, the Codex comparison, and deterministic script checks alike. Within-skill comparisons carry that pattern past differences in overall task difficulty, and the negative controls above mark how far a single-channel estimate can travel before it stops naming anything an author could act on.

H Comparison with Representative Benchmarks

Table 21 places SkillIF alongside the benchmark families discussed in Section 2. The comparison is restricted to what each reported evaluation protocol makes explicit, so an unmarked feature records one that the protocol does not state rather than one that its authors could not support.

Reusable skill packages distinguish a specification that is reused across assignments from instructions written for a single prompt. Environment execution covers evaluation inside a working environment with files, tools, and artifacts rather than on a response alone. Explicit adherence scoring means the protocol reports whether stated requirements were met, as opposed to only whether the task succeeded or failed. Hierarchical constraint scoring requires a nested taxonomy, so that scores can be read both at the level of a category of requirements and at the level of an individual subcategory within it. Across these four properties, the protocols we surveyed each leave some unstated: none combines reusable skill packages with reported adherence scoring, and none reports scores at two levels of a nested taxonomy.

Table 22 records the scale of each protocol and the mechanism it uses to judge compliance, with every figure taken from the source cited for that row. The rows are not ordered by scope, and a finer taxonomy does not by itself indicate broader coverage, because benchmarks partition their requirements along different lines.

I Skill Sources

Skills were collected from public repositories. More than 2,000 candidates were screened, and 96 skills were selected. Sources include calesthio/generative-media-skills, geogons/skill-financial-analyst, mastepanoski/claude-skills, mattpocock/skills, nicepkg/ai-workflow,

Table 22: **Comparison with instruction-following and skill-evaluation benchmarks.** Samples, constraint specification, evaluation focus, and judging mechanism as reported by each benchmark's authors.

Benchmark	Samples	Constraint specification	Evaluation focus	Eval. method
IFEval (Zhou et al., 2023)	541 prompts	25 types	Verifiable response constraints	Code
IFBench (Pyatkin et al., 2026)	300 prompts	58 templates	Generalization to unseen constraints	Code
FollowBench (Jiang et al., 2024)	820 instructions	5 categories + mixed	Increasing constraint complexity	Code + LLM
CFBench (Zhang et al., 2025)	1,000 prompts	10 categories; 25 subcategories	Realistic multi-constraint instructions	LLM
Multi-IF (He et al., 2024)	4,501 conversations	IFEval-derived constraints	Multi-turn, multilingual instruction following	Code
MultiChallenge (Deshpande et al., 2025)	273 conversations	Instance-level rubrics	Multi-turn conversational requirements	LLM
AgentIF (Qi et al., 2026)	707 instructions	8,415 constraints	Instruction following in agentic scenarios	Code + LLM
AgentIF-OneDay (Chen et al., 2026a)	104 tasks	767 scoring points	Task-level instruction following	LLM
SkillsBench (Li et al., 2026c)	87 tasks	Task-specific verifiers	Task-performance gains from supplied skills	Code
SkillIF (ours)	96 skill–task pairs	2,194 constraints; 3 categories / 9 subcategories	**Skill-specification adherence across three categories**	Code + LLM

Samples retain each benchmark's evaluation unit. Constraint types, templates, and individual criteria are distinct counting units. Code + LLM denotes the use of both judging mechanisms, with combination rules varying by benchmark. SkillsBench statistics follow arXiv v4.

OpenClaudia/openclaudia-skills, and tradermonty/claude-trading-skills. We thank the authors and maintainers of these repositories for making them publicly available.