SkillsBench: Benchmarking How Well Agent Skills Work Across Diverse Tasks

Xiangyi Li[1,*], Yimin Liu[1,2,*], Wenbo Chen[3,*,†], Bingran You[1,4,*], Zonglin Di[5,‡], Yifeng He[6,‡], Shenghan Zheng[7,‡], Kyoung Whan Choe[8,‡], Jiankai Sun[9,‡], Shuyi Wang[9,‡], Chujun Tao[14,‡], Binxu Li[10,‡], Xuandong Zhao[4,‡], Hejia Geng[11], Xiaojun Wu[35], Junwei Zhou[9], Xiaokun Chen[12], Hanwen Xing[13], Yubo Li[14], Qunhong Zeng[9], Di Wang[15], Yuanli Wang[16], Roey Ben Chaim[17], Penghao Jiang[18], Haotian Shen[9], Luyang Kong[9], Xinyi Liu[9], Runhui Wang[9], Xuanqing Liu[9], Jiachen Li[19], Xin Lan[20], Yueqian Lin[21], Wengao Ye[11], Junwei He[22], Songlin Li[12], Yue Zhang[23], Yipeng Gao[13], Yijiang Li[24], Ze Ma[25], Liqiang Jing[23], Tianyu Wang[9], Kaixin Li[9], Yiqi Xue[13], Haoran Lyu[9], Yizhuo He[14], Yuchen Tian[9], Shutong Wu[26], Bowei Wang[9], Yixuan Gao[27], Bo Chen[9], Litong Liu[28], Sikai Cheng[28], Jiajun Bao[14], Shuaicheng Tong[28], Shuwen Xu[9], Terry Yue Zhuo[9], Tinghan Ye[28], Qi Qi[9], Miao Li[28], Longtai Liao[9], Zelin Tan[34], Chang Shi[19], Xilin Tang[29], Srinath Tankasala[3,†], Boqin Yuan[24], Yaoyao Qian[30], Jianhong Tu[5], Chenguang Wang[5], Yizhou Sun[31], Wei Wang[31], Aaron Taylor[33], Ziyue Yang[6], Changkun Guan[28], Zhikang Dong[32], Xinyu Zhang[36], Steven Dillmann[12], Han-chung Lee[9], Dawn Song[4]

Abstract

Agent Skills are structured packages of procedural knowledge that augment large language model (LLM) agents at inference time. Despite rapid adoption, there is no standard way to measure whether they actually help. We present SKILLSBENCH, a benchmark whose current inventory contains 87 tasks across 8 domains paired with curated Skills and deterministic verifiers. Our latest aggregate evaluation runs the 87-task benchmark under matched no-Skills and curated-Skills conditions for 18 model–harness configurations. Curated Skills raise the average pass rate from 33.9% to 50.5% (+16.6 percentage points; 25.5% normalized gain), with configuration-level gains ranging from +4.1 to +25.7 pp. Focused Skills with at most three modules outperform larger or exhaustive bundles, and smaller models with Skills can match larger models without them. SKILLSBENCH establishes paired evaluation as the foundation for rigorous measurement of Skill efficacy on agentic, expertise-heavy work.

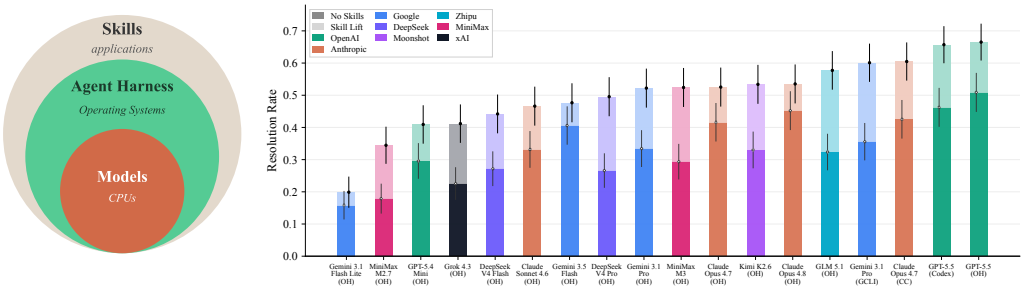

Figure 1: Agent architecture stack and resolution rates across 18 model–harness configurations on 87 SkillsBench tasks. Each bar stacks the no-Skills baseline and curated-Skills lift; open/filled interval markers denote 95% CIs for the baseline/total. Bars are ordered by Curated-Skills pass rate and colored by model family. Harness abbreviations: OH = OpenHands, CC = Claude Code, GCLI = Gemini CLI.

[1]BenchFlow, [2]OSU, [3]Amazon, [4]UC Berkeley, [5]UC Santa Cruz, [6]UC Davis, [7]Dartmouth, [8]RLWRLD, [9]Independent, [10]Princeton University, [11]Oxford University, [12]Stanford University, [13]USC, [14]CMU, [15]Foxconn, [16]BU, [17]Zenity, [18]UNSW, [19]UT Austin, [20]MSU, [21]Duke University, [22]ByteDance, [23]UT Dallas, [24]UC San Diego, [25]Columbia University, [26]University of Rochester, [27]Cornell Tech, [28]Georgia Tech, [29]Cornell University, [30]NEU, [31]UCLA, [32]Snap Inc., [33]Fanshawe College, [34]University of Science and Technology of China, [35]HKUST(GZ), [36]Anyscale

[*]Equal contribution. [‡]Core contribution. [†]This work was conducted outside the author's role at Amazon. Correspondence: Xiangyi Li at xiangyi@benchflow.ai.

1 Introduction

AI agents are now deployed in production workflows, from software engineering [Anthropic, 2025b, Google, 2025, OpenAI, 2025], where they sustain task horizons of up to 10 hours of human effort, to expertise-heavy domains beyond it. A fundamental tension follows: foundation models provide broad capabilities but lack the procedural knowledge a specialist brings on day one, while fine-tuning each domain is expensive and sacrifices generality [Brown et al., 2020, Ouyang et al., 2022, Yao et al., 2023b].

Agent Skills [Anthropic, 2025a] are an emerging solution: structured packages of instructions, code templates, resources, and reference material that augment agents at inference time without modifying model weights. Skills encode standard operating procedures, domain conventions, and task heuristics as modular artifacts mediated by the agent harness [Sutton et al., 1999, Sumers et al., 2024].

Community Skills ecosystems have already grown to 2,014,000 source-partitioned Skills in our construction snapshot (Figure 2; Appendix A). Yet despite this proliferation, no benchmark systematically asks: **how much do Skills actually help, and when do they fail?** Existing agent benchmarks [Liu et al., 2023, Merrill et al., 2026, Jimenez et al., 2024, Zhou et al., 2024b, Xie et al., 2024, Koh et al., 2024, Trivedi et al., 2024, Yang et al., 2023, Chan et al., 2025, Zhuo et al., 2025] measure raw capability in isolation, asking how well a model performs task X while folding model, harness, and augmentation effects into one pass rate. They do not answer the deployment question: *will adding this Skill help my agent on this task, and by how much?* Figure 1 illustrates the layered architecture and previews our main result: curated Skills improve resolution rates across 18 model–harness configurations by +16.6 pp on average.

We introduce SKILLSBENCH, the first benchmark that treats Skills as a first-class evaluation artifact. Our contribution is two-tier:

- **Narrow contribution: a quantitative answer to "how much do Skills help?"** On the 87-task benchmark, evaluated under matched no-Skills and curated-Skills conditions across 18 model–harness configurations, curated Skills lift task-macro pass rate from 33.9% to 50.5% (+16.6 pp; 25.5% normalized gain), with substantial configuration-level heterogeneity (+4.1 to +25.7 pp).
- **Broad contribution: a paired-evaluation framework for agent augmentation, generalizable beyond Skills.** The same paired (with vs. without) protocol, contributor-driven sourcing, leakage-controlled task-to-artifact decoupling, and BenchFlow [BenchFlow team, 2026] containerized harness can evaluate other artifacts (retrieval pipelines, memory stores, scaffolding) without confounding model and augmentation effects. We open-source the benchmark, harness, and public trajectories/results so practitioners can test their own Skill libraries before shipping.

2 Background

Skill Definition In this paper, we define a **Skill** as a reusable, file-system-based procedural package for a class of agent tasks. Each Skill contains a required `SKILL.md` file with natural-language instructions and may optionally include auxiliary resources such as scripts, templates, reference files, or worked examples.

Skill Augmentation We use **Skill augmentation** to refer to the inference-time mechanism by which an agent harness makes relevant Skills available to an agent for solving a task. Compared with other runtime augmentation paradigms, Skill augmentation is *modular and reusable*, provides *procedural guidance* rather than factual context alone, can include *executable resources*, and is *cross-model portable* because Skills are represented as files rather than model parameters (Table 1).

Agent Harness An **agent harness** is the execution layer that wraps an LLM and connects it to its environment [Lopopolo, 2026, Lee, 2026]. It exposes tools, manages files and workspace state, executes scripts, returns observations, and, in Skill-augmented agents, discovers and loads relevant Skills during inference.

Thus, a Skill is neither model weights nor an executable tool by itself; it becomes actionable through a compatible harness that injects its instructions and exposes its resources.

Table 1: Comparison of runtime augmentation paradigms. Skills combine modular packaging, procedural guidance, executable resources, and portability.

	Prompts	**RAG**	**Tools**	**Skills**
Modular/reusable	×	✓	✓	✓
Procedural guidance	Limited	×	×	✓
Executable resources	×	×	✓	✓
Cross-model portable	✓	✓	✓	✓

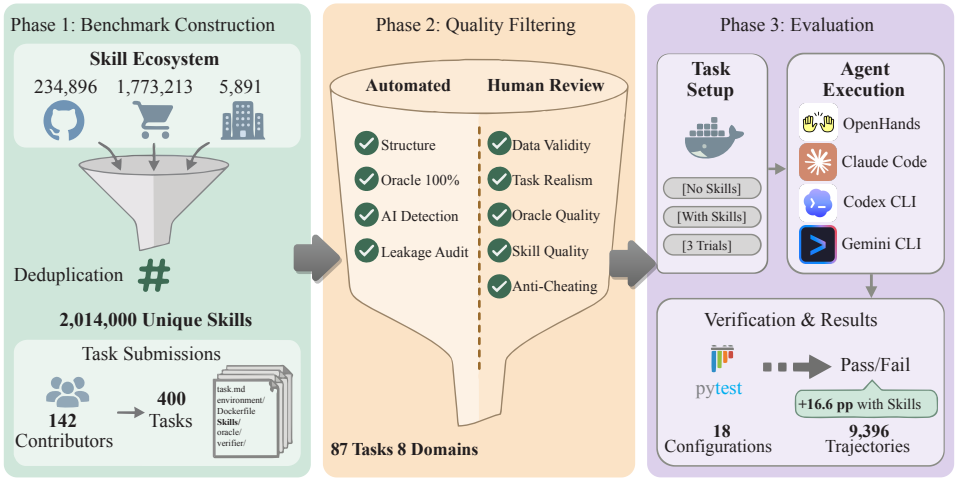

Figure 2: **SKILLSBENCH pipeline overview.** Construction aggregates 2,014,000 source-partitioned Skills and 400 candidate task submissions from 142 contributors. Filtering applies automated checks and human review, retaining an 87-task public evaluation aggregate across 8 domains. Evaluation runs each task under matched Skill-access conditions across 18 model–harness configurations on BenchFlow [BenchFlow team, 2026], producing 9,396 selected public result files.

3 SKILLSBENCH

A benchmark that measures Skill efficacy is only as credible as its filtering pipeline: if a Skill can encode task-specific answers, the measurement collapses into instruction-following. SKILLSBENCH therefore treats construction as part of the contribution. We define valid tasks, source candidates from a vetted contributor community, and filter them through automated gates and human review. The current evaluated inventory contains 87 tasks across 8 domains, drawn from 400 submissions by 142 contributors. Each task is a containerized unit with fixed data, an oracle solution, and a deterministic verifier, evaluated under matched no-Skills and Skills-augmented conditions to isolate the Skill's contribution from the model–harness configuration.

Skills as expertise, not answers. SKILLSBENCH requires Skills to provide domain expertise for a *class* of problems, never the solution to a specific instance. We enforce this in two ways. First, contributors author Skills independently of the benchmark—from public repositories or prior domain experience—so the experiment measures use of pre-existing expertise. Second, task instructions never name which Skills to use; agents discover and activate Skills through the standard progressive-disclosure mechanism [Anthropic, 2025a]. Without these constraints, a Skill becomes a hidden answer key.

Task principles. Beyond the Skill boundary, every task must be *authentic* real work, *verifiable* by deterministic pass/fail tests rather than LLM-as-a-judge [Wang et al., 2023b, Brown, 2025], *difficult* because of the problem rather than artificial instruction confusion, and *solvable* end-to-end by an oracle agent without pre-baked answers or hard-coded magic numbers. Instructions are human-authored; if LLMs help refine wording, a human owns the final iteration. We reject classroom-style tasks, made-up scenarios, toy datasets, and purely synthetic data.

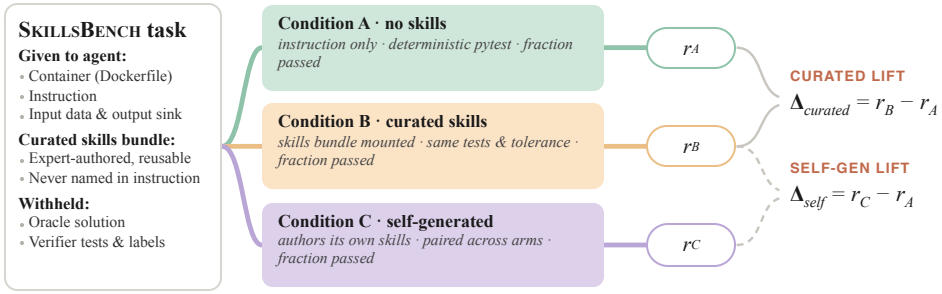

Figure 3: **Anatomy of a SKILLSBENCH task.** Each task ships with a containerized environment, human-authored instruction, expert-curated Skills bundle, and withheld oracle/verifier/labels. The arms differ only in Skill access (A: instruction alone; B: curated bundle mounted for the agent to discover; C: the agent authors its own skill documents, then solves with them); one deterministic `pytest` battery scores every arm as the fraction of checks passed, $r_A, r_B, r_C \in [0, 1]$. Skill efficacy: $\Delta_{\text{curated}} = r_B - r_A$, and $\Delta_{\text{self}} = r_C - r_A$ against the same baseline.

Sourcing and filtering. We grew a 1,400-member contributor community, onboarded 200+ task authors through scoping interviews, and ingested 400 candidate submissions from the public PR history. Every PR clears four automated gates before human review: structural integrity, oracle execution, instruction provenance (AI-text detector plus human label, yielding a 100% human-authored release set), and Skill-to-solution leakage (rejecting task-specific filenames, paths, magic numbers, verbatim oracle commands, or expected outputs). PRs that pass receive at least one 30-minute maintainer review; tasks with no measurable separation between conditions are rejected as low-signal. The current public evaluation aggregate retains 87 tasks; per-gate rejection counts and the full rubric are in Appendix M.

Task specification. A SKILLSBENCH task is a self-contained module with four components (Figure 3), following the Agent Skills format [Anthropic, 2025a]: a human-authored *instruction*, a Docker *environment* with task data and a `skills/` subdirectory, an *oracle* reference solution that must pass the verifier, and a deterministic `pytest` *verifier*. Each Skill folder uses the standard layout: a required `SKILL.md` plus optional scripts, references, and assets.

Composition. The current task inventory contains 87 tasks across 8 domains (Figure 4), stratified by estimated human-specialist completion time without AI assistance: 6 Core (<60 min), 53 Extended (1–4 h), 28 Extreme (>4 h). Per-domain counts range from $N = 5$ to $N = 16$.

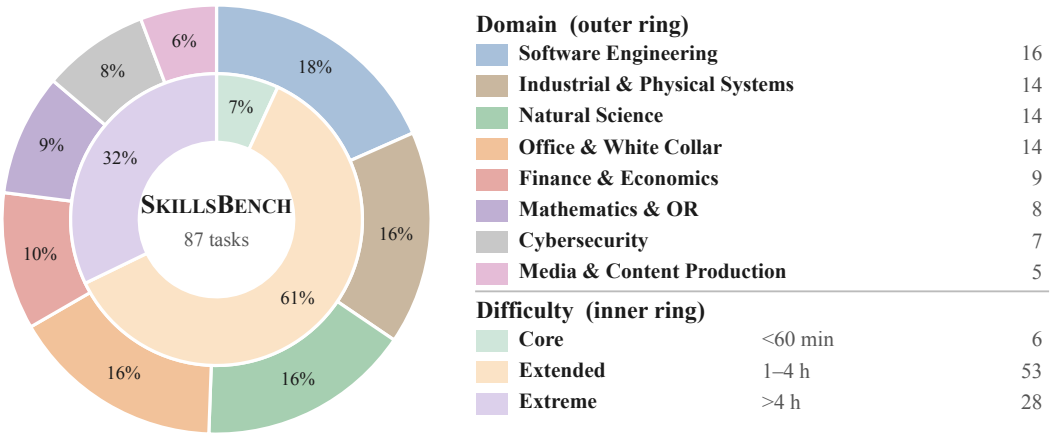

Figure 4: SKILLSBENCH spans 8 domains (87 tasks total), weighted toward production workflows rather than classroom problem sets. The inner ring shows the difficulty stratification; domain definitions and the per-task mapping appear in Table 19.

4 Experiment

Models, harnesses, and conditions. For the latest aggregate in Table 2, we evaluate 18 configurations across four terminal-agent harnesses: OpenHands, Gemini CLI [Google, 2025], Claude Code [Anthropic, 2025b], and Codex CLI [OpenAI, 2025]. Each task runs under two matched conditions: *no Skills* (the task instruction only) and *curated Skills* (the task's full `environment/skills/` directory). A third *self-generated Skills* condition—the agent first authors skill packs with Anthropic's `skill-creator`, then solves with only those packs—is evaluated on the three dedicated-harness configurations and reported separately (Appendix D.6).

Evaluation harness and protocol. We instantiate SKILLSBENCH on **BenchFlow** [BenchFlow team, 2026], an open-source multi-turn agent benchmarking framework with uniform Skill injection, sandboxing, and trajectory logging. For each (configuration, task, condition) triple, BenchFlow builds a fresh pinned container, hands the agent the task instruction (and Skills under the relevant condition), and runs until final submission. The deterministic `test-script` verifier then emits a pass/fail result. Unscored, stale, or rate-limited trajectories are treated as incomplete coverage for rerun/audit; timeout rows are used only when healthy pass/fail replacements are unavailable and then scored as failures. The latest aggregate targets three selected public trials per (configuration, task, condition) cell against the fixed 87×3 frame.

Metrics. The primary metric is task-macro pass rate, following Terminal-Bench [Merrill et al., 2026]: per-task pass/fail outcomes are averaged over the three-trial frame, then across the fixed 87-task inventory. We complement this with normalized gain [Hake, 1998]

$$g \;=\; (\text{pass}_{\text{skill}} - \text{pass}_{\text{vanilla}}) \,/\, (1 - \text{pass}_{\text{vanilla}}), \tag{1}$$

which measures the fraction of remaining headroom Skills close. Because every condition runs on the same task in the same container, deltas are paired differences at the (configuration, task) level, not unpaired-pool differences. Aggregation rules and per-cell trial counts are in Appendix N. For rows that summarize multiple configurations, we compute g separately for each configuration at full precision and macro-average those normalized gains, so the reported mean g can differ slightly from recomputing g from mean pass rates.

5 Results

We report aggregate no-Skills vs. curated-Skills efficacy, then analyze Skills design factors, domain effects, and task-level failures.

5.1 Skills Efficacy Across Model–Harness Combinations

We evaluate each commercial model–harness configuration under matched no-Skills and curated-Skills conditions on all 87 tasks.

5.1.1 Main Results

Table 2 reports pass rates for each model–harness combination, ordered by with-Skills performance.

Finding 1: Skills provide broad but non-uniform gains. Every one of the 18 model–harness configurations improves with curated Skills (Figure 1). The average gain is +16.6 pp, but the spread is large (+4.1 to +25.7 pp): most configurations see double-digit gains, while OpenHands + Gemini 3.1 Flash Lite is nearly flat (+4.1 pp). Skill efficacy is therefore an empirical property of a specific agent stack rather than a universal constant.

Finding 2: The strongest absolute systems are not always the largest beneficiaries. The highest with-Skills pass rates are OpenHands + GPT-5.5 (67.3%), Codex + GPT-5.5 (66.5%), and Claude Code + Opus 4.7 (61.2%). The largest lifts instead come from OpenHands + GLM 5.1 (+25.7 pp; 38.1% normalized gain), Gemini CLI + Gemini 3.1 Pro (+24.8 pp), and OpenHands + DeepSeek V4 Pro (+23.2 pp). Conversely, OpenHands + Gemini 3.5 Flash starts from a strong no-Skills baseline (41.1%) but gains only +7.1 pp, showing that high base capability does not imply high Skill leverage.

Table 2: Pass rates (%), absolute gain (Δ, pp), and normalized gain (g, %) across the latest 87-task no-Skills vs. curated-Skills aggregate. Δ and g are computed at full precision; configurations are ordered by Curated-Skills pass rate. The Mean-row g macro-averages per-configuration normalized gains rather than recomputing g from the mean pass rates. Each pass cell uses the fixed 87×3 trial frame; per-cell public coverage is reported in Appendix I. Differences from rounded *Pass* cells may differ by ± 0.1 pp.

		No Sk.	**Curated Skills**		
Harness	**Model**	**Pass**	**Pass**	**Δ (pp)**	**g (%)**
OpenHands	GPT-5.5	51.5	67.3	+15.8	32.6
Codex	GPT-5.5	46.8	66.5	+19.7	37.0
Claude Code	Opus 4.7	43.0	61.2	+18.2	31.9
Gemini CLI	Gemini 3.1 Pro	36.0	60.8	+24.8	38.7
OpenHands	GLM 5.1	32.7	58.4	+25.7	38.1
OpenHands	Claude Opus 4.8	45.7	54.1	+8.4	15.5
OpenHands	Kimi K2.6	33.4	54.0	+20.6	31.0
OpenHands	Claude Opus 4.7	42.1	53.1	+11.1	19.1
OpenHands	MiniMax M3	29.7	53.0	+23.3	33.2
OpenHands	Gemini 3.1 Pro	33.8	52.8	+19.0	28.7
OpenHands	DeepSeek V4 Pro	26.9	50.1	+23.2	31.8
OpenHands	Gemini 3.5 Flash	41.1	48.2	+7.1	12.1
OpenHands	Claude Sonnet 4.6	33.5	47.2	+13.6	20.5
OpenHands	DeepSeek V4 Flash	27.5	44.7	+17.2	23.7
OpenHands	Grok 4.3	22.8	41.7	+18.8	24.4
OpenHands	GPT-5.4 Mini	29.9	41.4	+11.5	16.4
OpenHands	MiniMax M2.7	18.1	34.9	+16.8	20.5
OpenHands	Gemini 3.1 Flash Lite	16.0	20.1	+4.1	4.9
Mean		33.9	50.5	**+16.6**	25.5

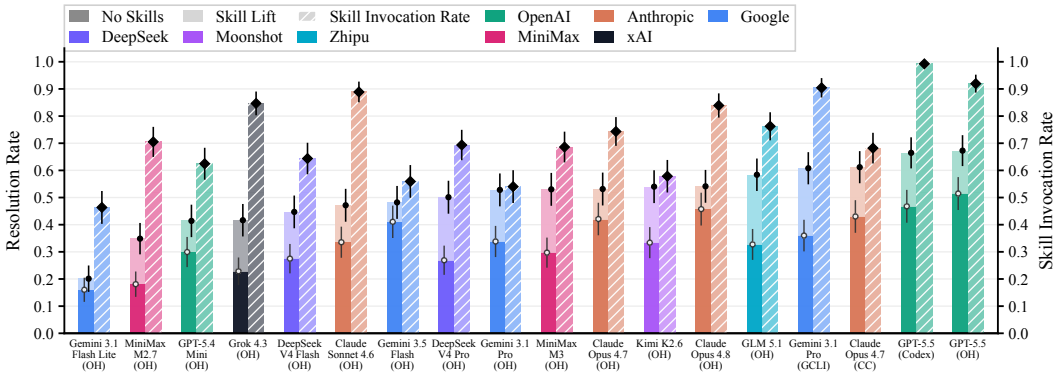

Figure 5: Task-specific Skill Invocation Rate alongside resolution rates. Left axis: stacked resolution-rate bars (no-Skills baseline plus curated-Skills lift). Right axis: adjacent hatched Skill Invocation Rate bars. Open/filled circles mark 95% CIs for no-Skills/with-Skills resolution rates; black diamonds mark invocation-rate CIs.

Finding 3: Harness choice materially changes how the same model uses Skills. Fig. 1 includes three model families evaluated under multiple harnesses. GPT-5.5 reaches 67.3% with Skills in OpenHands and 66.5% in Codex; Gemini 3.1 Pro reaches 60.8% in Gemini CLI but 52.8% in OpenHands; Claude Opus 4.7 reaches 61.2% in Claude Code and 53.1% in OpenHands. Skills are therefore not merely extra context: the harness's discovery, prompting, execution, and tool-use loop mediate realized benefit. Appendix I reports the per-configuration public-trial counts.

Finding 4: Skill discovery is usually not the bottleneck. Figure 5 separates harness-bundled Skills, treated as part of the harness, from task-specific Skills shipped with each benchmark task. We count only the latter: a curated-Skills trial counts when its trajectory reads or invokes a Skill under

Table 3: Skills efficacy (%) by domain across the latest 87-task benchmark set. N is the number of tasks per domain. Pass rates use a fixed three-trial denominator within each (condition, config, task), then macro-average across tasks and 18 model–harness configurations; all domains show positive aggregate delta.

Domain	N	No Skills	With Skills	Δ_{abs}
Natural Science	14	42.0%	70.8%	+28.8
Media & Content Production	5	23.3%	47.4%	+24.1
Cybersecurity	7	29.5%	48.4%	+18.9
Industrial & Physical Systems	14	23.9%	39.6%	+15.7
Finance & Economics	9	19.1%	33.3%	+14.2
Office & White Collar	14	40.5%	53.0%	+12.6
Software Engineering	16	37.6%	49.2%	+11.6
Mathematics & OR	8	45.7%	55.4%	+9.7

that task's `environment/skills`. High Skill Invocation Rate does not guarantee high resolution, so the remaining failures often occur after task-Skill access. Exact counts are in Appendix K.

Self-generated Skills do not substitute for curated ones. On the three dedicated-harness configurations we additionally evaluate a self-generated condition: the agent first authors skill packs with Anthropic's `skill-creator`, then solves each task with only those packs (Appendix D.6). Self-generated Skills land *below* the no-Skills baseline on all three configurations (-8.1 pp on Claude Code + Opus 4.7, -11.3 pp on Codex + GPT-5.5, -11.5 pp on Gemini CLI + Gemini 3.1 Pro), while curated Skills add $+18.2$ to $+24.8$ pp on the same configurations. A trajectory audit attributes the deficit to generated packs the solver never discovers, creator-side authoring that displaces solver work, and confidently wrong pack content when the skills are used (Appendix D.6.1).

5.1.2 Domain-Level Analysis

Finding 5: Skills benefit varies widely across domains. Table 3 shows substantial heterogeneity. Natural Science (+28.8 pp), Media & Content Production (+24.1 pp), and Cybersecurity (+18.9 pp) benefit most; Software Engineering (+11.6 pp) and Mathematics & OR (+9.7 pp) benefit least. Domains requiring specialized procedural knowledge underrepresented in model pretraining (e.g., scientific signal processing, security analysis, and multimedia transformation workflows) improve most, whereas domains with stronger pretraining and tooling coverage benefit less from external procedural guidance.

5.1.3 Task-Level Analysis

Task-level results reveal high variance in Skills effectiveness:

Top Skills beneficiaries. The 10 highest-Δ tasks span prefix-cache replay, intrusion detection, SEC filings, wet-lab analysis, hydrology, geoscience, 3D parsing, optimization, and dependency auditing, with mean improvement $+67.0$ pp. `llm-prefix-cache-replay` reaches 94.4% from a 1.9% no-Skills pass rate, and `dapt-intrusion-detection` reaches 81.5% from zero (full list in Appendix J.3; Skill content audit in Appendix F.4).

Failure modes when Skills hurt performance. 13 of 87 tasks show negative Skills deltas; the largest drops occur on `exam-block-sequencing` (-7.4 pp), `suricata-custom-exfil` (-7.4 pp), and `adaptive-cruise-control` / `mars-clouds-clustering` / `r2r-mpc-control` / `econ-detrending-correlation` (-5.6 pp each). Paired-trajectory audit (Appendix F.3) surfaces three repeatable patterns: the Skill prescribes an unnecessarily heavyweight pipeline, displaces a stronger default strategy, or points the agent at a solver it cannot debug. The common cause is a single "correct" pipeline without applicability boundaries or lightweight fallbacks.

Finding 6: Compact, focused Skills outperform exhaustive ones, and small models with Skills can match larger models without. Two design ablations (Appendix F) refine the picture. First, Skill *quantity*: tasks paired with one Skill gain +18.0 pp, 2–3 Skills gain +19.0 pp, and ≥ 4 Skills

give only +10.1 pp, suggesting excess content creates overhead or conflicting guidance. Second, Skill *complexity*: compact and standard-length Skills (+19.0 and +21.5 pp) outperform detailed (+14.5 pp) and comprehensive documentation (+0.7 pp); focused procedural guidance beats exhaustive prose. Finally, scale: MiniMax M2.7 with Skills (34.9%) exceeds stronger no-Skills baselines such as OpenHands + GLM 5.1 (32.7%) and OpenHands + MiniMax M3 (29.7%), so Skills can partly compensate for model capacity on procedural tasks.

6 Discussion

Skills close procedural gaps. Skills are most helpful when success depends on concrete procedures and verifier-facing details (steps, constraints, sanity checks), rather than broad conceptual knowledge. Gains are largest for specialized workflows or brittle formats, and smaller or negative when models already have strong priors or the Skill adds overhead.

Harnesses mediate Skills use. Skills efficacy depends not only on Skills quality but also on harness implementation. Some harnesses reliably retrieve and use Skills, while others acknowledge Skills content but proceed without invoking it. Structured interfaces can also introduce long-trajectory failure modes (e.g., format drift), reducing the influence of early-injected Skills. This motivates evaluating Skills under multiple harnesses rather than treating "with Skills" as a single condition.

Implications for Skill authoring. Trajectory audit on the 10 highest-Δ tasks (Appendix F.4) yields five recurring patterns: executable scripts with calibrated defaults, canonical data sources and parsing quirks, verifier-facing file-format constraints, algorithmic invariants, and task-specific `description:` frontmatter for first-scan matching. Comprehensive prose is essentially flat while focused documentation yields larger aggregate lift ($+0.7$ pp vs. +19.0/+21.5 pp; Appendix F); Skill authors should optimize for verifier-facing detail an agent cannot infer, not for completeness.

Complexity-aware fallback paths for Skills. The three failure modes that drive negative deltas (Appendix F.3) all share the same root cause: the Skill prescribes a pipeline that is correct in principle but too heavy or brittle for routine agent execution. We propose extending the `SKILL.md` frontmatter with an explicit complexity contract: expected tool and token cost, applicability boundaries, and a required lightweight fallback path. This would help agents route around heavy recipes when the task evidence does not justify them, and it would force Skill authors to state when their "correct" workflow assumes unusually expensive computation.

6.1 Limitations and Future Work

Coverage and generalization. SKILLSBENCH focuses on terminal-based, containerized tasks for reproducible evaluation, so results may not transfer directly to GUI agents, multi-agent coordination, or very long-horizon workflows. We also evaluate a limited set of models and harnesses whose Skills integration can change over time. A natural extension is multi-modal Skills and protocols for vision-language agents in GUI environments.

Causal attribution and controls. Skills injection increases context length, so observed gains could partly reflect "more context" rather than procedural structure. Our self-generated Skills condition suggests that model-authored procedural text does not reproduce the curated-Skills gain (Appendix D.6), though its deficit mixes content quality with skill-discovery and creator/solver interference effects. Future work requires stronger length-matched baselines, such as random or irrelevant text and retrieval-only documentation, to isolate which components (steps, examples, code resources) drive improvement and to study automatic Skills synthesis.

Determinism, contamination, and ecological validity. Containerization provides state isolation but not perfect determinism or immunity to training-set leakage. We mitigate with multiple runs, a leakage audit (§3), and paired (Skills vs. no Skills) comparisons, yet cannot eliminate all nondeterminism or memorization effects. Future work should evaluate ecosystem-representative settings, including lower-quality and automatically selected Skills, and study Skills composition: when multiple Skills help or interfere, and whether composite performance can be predicted from atomic effects.

7 Related Work

We situate SKILLSBENCH within agent benchmarks, procedural augmentation, and evaluation methodology.

Agent benchmarks. Recent benchmarks evaluate end-to-end agent capability across realistic environments, including Terminal-Bench [Merrill et al., 2026], SWE-bench and follow-ons [Jimenez et al., 2024, Yang et al., 2024, 2025], AgentBench and web/GUI settings [Liu et al., 2023, Zhou et al., 2024b, Koh et al., 2024, Xie et al., 2024], and suites for tool-mediated workflows, execution feedback, or domain specialization [Yao et al., 2025, Trivedi et al., 2024, Yang et al., 2023, Chan et al., 2025, Zhang et al., 2024, Zhuo et al., 2025, Austin et al., 2021, Ye et al., 2025]. These benchmarks measure fixed-agent task completion. SKILLSBENCH instead measures *augmentation efficacy* via paired evaluation.

Procedural augmentation and tool use. Prior work augments agents with structured reasoning or external knowledge, including CoALA and Voyager [Sumers et al., 2024, Wang et al., 2023a], multi-step reasoning methods [Wei et al., 2022, Yao et al., 2023a,b, Shinn et al., 2023, Madaan et al., 2023, Zhou et al., 2023a, 2024a], retrieval and tool use [Lewis et al., 2020, Zhou et al., 2023b, Schick et al., 2023, Qin et al., 2024], and declarative optimization frameworks [Khattab et al., 2023]. Skills combine procedural guidance with executable resources (§2); SKILLSBENCH quantifies their actual impact.

Skills ecosystems and evaluation methodology. Anthropic's Agent Skills and MCP specifications [Anthropic, 2025a, 2024] formalized Skill packages and tool connectivity, and agent CLIs provide real-world harnesses [Anthropic, 2025b, Google, 2025, OpenAI, 2025]. SKILLSBENCH runs on BenchFlow [BenchFlow team, 2026] and uses Terminal-Bench scoring [Merrill et al., 2026] for comparability.

Paired evaluation of agent augmentations. The closest methodological precedent is paired evaluation of inference-time augmentations on enterprise API workflows, which contrasts with-vs.-without conditions on the same task and reports paired bootstrap CIs [Liu et al., 2023]. SKILLSBENCH adopts this paired-condition design while decoupling tasks from independently authored Skills and spanning 8 expertise-heavy domains rather than a single workflow class. Broader benchmarking practice motivates careful reporting and comparability [Mattson et al., 2020, Chiang et al., 2024, Srivastava et al., 2023]; we therefore report both absolute gains and normalized gain [Hake, 1998] across No-Skills baselines (§4).

8 Conclusion

We introduced SKILLSBENCH, a paired benchmark for evaluating Agent Skills as first-class artifacts. Across the latest 87-task, 18-configuration no-Skills vs. curated-Skills aggregate, curated Skills improve performance substantially but unevenly (+16.6 pp on average; range +4.1 pp to +25.7 pp by configuration). Concise procedural Skills outperform exhaustive documentation, and Skills can partially substitute for model scale on procedural tasks. We release the benchmark, the BenchFlow harness, and the public trajectories/results.

Acknowledgments

We thank the organizations whose models we evaluate—OpenAI, Anthropic, Google, Z.ai, Moonshot AI, DeepSeek, xAI, MiniMax, Alibaba Qwen, Tencent Hunyuan, and Xiaomi MiMo—for model access and API support. We thank Google Cloud Platform and Amazon Bedrock for cloud and inference infrastructure support, and the OpenHands team for the open-source harness used in our main aggregate. We thank Junxian He for guidance and advice; Yi R. (May) Fung for advice and paper-writing guidance; Gabriel Chua for providing last-minute API keys; Zachary Mueller for providing GPU resources through Lambda; and Ivan Burazin for providing Daytona credits. We also thank the following contributors for task and code contributions: Jianheng Hou, Jierun Chen, Jiahao Liu, Shutong Wu, Yulin Li, Zhenheng Tang, Benny Jiang, Qingyang Ma, Issa Sugiura, Jinya Jiang, Connor Adams, and Ananth Jayakrishnan Nambiar; and all open-source contributors to the SKILLSBENCH repositories.

References

Anthropic. Introducing the model context protocol. https://www.anthropic.com/news/model-context-protocol, November 2024.

Anthropic. Equipping agents for the real world with Agent Skills. https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills, October 2025a.

Anthropic. Claude Code: an agentic coding tool. https://github.com/anthropics/claude-code, 2025b.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

BenchFlow team. BenchFlow: framework for RL environments for LLM agents. https://github.com/benchflow-ai/benchflow, 2026.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *NeurIPS*, 2020.

William Brown. Verifiers: Environments for llm reinforcement learning. https://github.com/PrimeIntellect-ai/verifiers, 2025.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Aleksander Madry, and Lilian Weng. MLE-bench: Evaluating machine learning agents on machine learning engineering. In *ICLR*, 2025.

Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Banghua Zhu, Hao Zhang, Michael Jordan, Joseph E Gonzalez, et al. Chatbot arena: An open platform for evaluating llms by human preference. In *ICML*, 2024.

Google. Gemini CLI: An open-source AI agent that brings the power of Gemini directly into your terminal. https://github.com/google-gemini/gemini-cli, 2025.

Richard R Hake. Interactive-engagement versus traditional methods: A six-thousand-student survey of mechanics test data for introductory physics courses. *American journal of Physics*, 66(1):64–74, 1998. ISSN 0002-9505. doi: 10.1119/1.18809.

Harbor Framework Team. Harbor: A framework for evaluating and optimizing agents and models in container environments. https://github.com/harbor-framework/harbor, 2026.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *ICLR*, 2024.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, et al. Dspy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*, 2023.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Russ Salakhutdinov, and Daniel Fried. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. In *ACL*, pages 881–905, 2024.

Hanchung Lee. A Taxonomy of RL Environments for LLM Agents. https://leehanchung.github.io/blogs/2026/03/21/rl-environments-for-llm-agents/, March 2026.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *NeurIPS*, 2020.

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. *arXiv preprint arXiv:2308.03688*, 2023.

Ryan Lopopolo. Harness engineering: leveraging Codex in an agent-first world. `https://openai.com/index/harness-engineering/`, February 2026.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *NeurIPS*, 2023.

Peter Mattson, Christine Cheng, Gregory Diamos, Cody Coleman, Paulius Micikevicius, David Patterson, Hanlin Tang, Gu-Yeon Wei, Peter Bailis, Victor Bittorf, et al. Mlperf training benchmark. *MLSys*, 2020.

Mike A Merrill, Alexander G Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E Kelly Buchanan, et al. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. *arXiv preprint arXiv:2601.11868*, 2026.

OpenAI. Codex CLI: Lightweight coding agent that runs in your terminal. `https://github.com/openai/codex`, 2025.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *NeurIPS*, 2022.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, dahai li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs. In *ICLR*, 2024.

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. *NeurIPS*, 2023.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *NeurIPS*, 2023.

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, et al. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *TMLR*, 2023.

Theodore Sumers, Shunyu Yao, Karthik R Narasimhan, and Thomas L Griffiths. Cognitive architectures for language agents. *TMLR*, 2024.

Richard S Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 1999.

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 *ACL*, 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*, 2023a.

Zhiruo Wang, Shuyan Zhou, Daniel Fried, and Graham Neubig. Execution-based evaluation for open-domain code generation. In *Findings of EMNLP 2023*, 2023b.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *NeurIPS*, 2022.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *NeurIPS*, 2024.

John Yang, Akshara Prabhakar, Karthik Narasimhan, and Shunyu Yao. Intercode: Standardizing and benchmarking interactive coding with execution feedback. *NeurIPS*, 2023.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *NeurIPS*, 2024.

John Yang, Kilian Lieret, Carlos E. Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. Swe-smith: Scaling data for software engineering agents. In *NeurIPS*, 2025.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *NeurIPS*, 2023a.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *ICLR*, 2023b.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik R Narasimhan. τ-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains. In *ICLR*, 2025.

Christine Ye, Sihan Yuan, Suchetha Cooray, Steven Dillmann, Ian LV Roque, Dalya Baron, Philipp Frank, Sergio Martin-Alvarez, Nolan Koblischke, Frank J Qu, et al. ReplicationBench: Can AI Agents Replicate Astrophysics Research Papers? *arXiv preprint arXiv:2510.24591*, 2025.

Andy K Zhang, Neil Perry, Riya Dulepet, Joey Ji, Celeste Menders, Justin W Lin, Eliot Jones, Gashon Hussein, Samantha Liu, Donovan Jasper, et al. Cybench: A framework for evaluating cybersecurity capabilities and risks of language models. *arXiv preprint arXiv:2408.08926*, 2024.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language Agent Tree Search Unifies Reasoning Acting and Planning in Language Models. In *ICML*, 2024a.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, and Ed Chi. Least-to-Most Prompting Enables Complex Reasoning in Large Language Models. In *ICLR*, 2023a.

Shuyan Zhou, Uri Alon, Frank F Xu, Zhengbao Jiang, and Graham Neubig. Docprompting: Generating code by retrieving the docs. In *ICLR*, 2023b.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A Realistic Web Environment for Building Autonomous Agents. In *ICLR*, 2024b.

Terry Yue Zhuo, Vu Minh Chien, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, Simon Brunner, Chen GONG, James Hoang, Armel Randy Zebaze, Xiaoheng Hong, Wen-Ding Li, Jean Kaddour, Ming Xu, Zhihan Zhang, Prateek Yadav, Naman Jain, Alex Gu, Zhoujun Cheng, Jiawei Liu, Qian Liu, Zijian Wang, David Lo, Binyuan Hui, Niklas Muennighoff, Daniel Fried, Xiaoning Du, Harm de Vries, and Leandro Von Werra. BigCodeBench: Benchmarking Code Generation with Diverse Function Calls and Complex Instructions. In *ICLR*, 2025.

A Skill Ecosystem Analysis

To contextualize SKILLSBENCH within the broader landscape of agent augmentation, we analyze the same construction snapshot reported in Figure 2: 234,896 direct GitHub/source Skills, 1,773,213 marketplace Skills, and 5,891 partner Skills, yielding 2,014,000 source-partitioned Skills (deduplicated within each source, then summed across partitions). When an analysis requires repository-level timestamps or cloned files, we report the reachable sample size explicitly while keeping 2,014,000 as the ecosystem-wide count.

A.1 Data Collection

We aggregate Skills from direct GitHub/source discovery, marketplace records, and partner inventories. Each entry records `name`, `description`, `author`, `url`, source partition, and, when available, the upstream repository `pushed_at` timestamp. Summed across these source partitions (after deduplicating within each), the construction snapshot contains 2,014,000 source-partitioned Skills. This is a renewal of an earlier 37k-Skill snapshot taken in January 2026; the corpus has grown by more than an order of magnitude in the intervening months.

For per-Skill structural statistics (size, file count, extension mix), we further cloned 767,430 Skill bundles whose source repositories were still reachable; the remaining entries either point to deleted/private repositories, partner/marketplace records without cloneable source, or repositories that failed to fetch within retry limits.

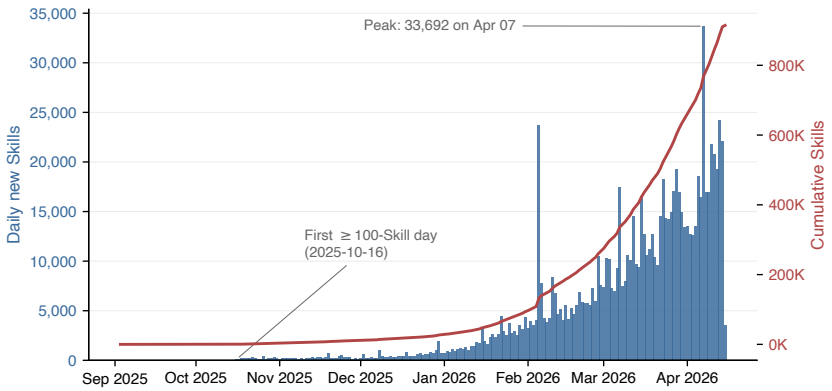

Figure 6: Temporal dynamics of Skill creation in the timestamped portion of the 2,014,000-Skill construction snapshot. The window starts at 2025-09-03 to provide a pre-launch baseline. Daily additions (bars, left axis) climb through late 2025 and surge through Q1 2026, peaking at 33,692 Skills on 2026-04-07. The first day with $\geq$100 new Skills is 2025-10-16; the 153 Skills with timestamps before that date are likely artifacts of `pushed_at` reflecting the upstream repo's last push (which can be backdated or simply inherited from a stale repo that later received a SKILL.md) rather than the Skill's actual creation time, so the pre-launch tail should not be read as evidence of real activity before the launch.

A.2 Skill Characteristics

Size Distribution. Skill sizes follow a heavy-tailed log-normal distribution (Figure 7). SKILL.md instructions (panel a) have a median of 4.8 KB ($\sim$1.2k tokens at 4 bytes/token; IQR 2.4–9.2 KB), while total bundle size (panel b) has a median of 7.2 KB ($\sim$1.8k tokens; IQR 3.0–17.4 KB). The tail is dramatic: the largest Skill bundle exceeds 1.6 GB.

Domain Coverage. Skills span 12 marketplace-assigned categories with broad coverage and no single dominant area (Figure 8; n=1,031,651 category assignments, where a Skill may carry multiple tags):

- Tools: 22.4% (CLIs, terminal utilities, build tools)

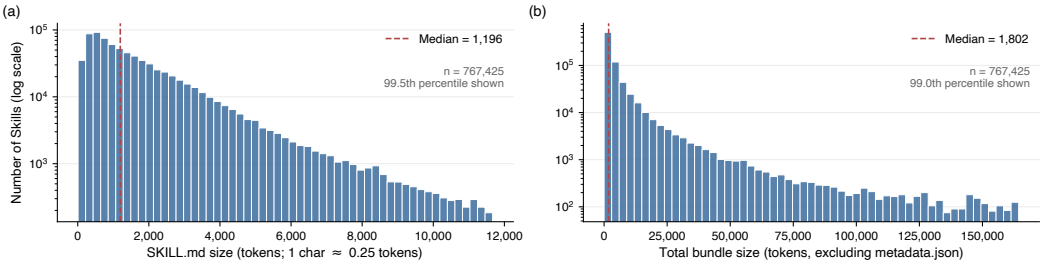

Figure 7: Skill size distributions in the reachable-clone sample (n=767,425; tokens approximated as bytes/4). **(a)** SKILL.md instructions, 99.5th percentile shown; median ∼1.2k tokens. **(b)** Total bundle size including all resources (excluding `metadata.json`), 99th percentile shown; median ∼1.8k tokens. Both distributions are highly skewed toward concise artifacts but the bundle-size tail extends several orders of magnitude.

- Business: 17.0% (workflows, productivity, ops)
- Development: 14.3% (general software engineering)
- Testing & Security: 9.8% (test scaffolds, security scans)
- Data & AI: 9.2% (data pipelines, ML, analytics)
- DevOps: 7.8% (Docker, Kubernetes, CI/CD)
- Documentation: 6.6% (technical writing, API docs)
- Content & Media: 6.0% (writing, design, media)
- Long tail: 6.9% (Research, Lifestyle, Databases, Blockchain combined)

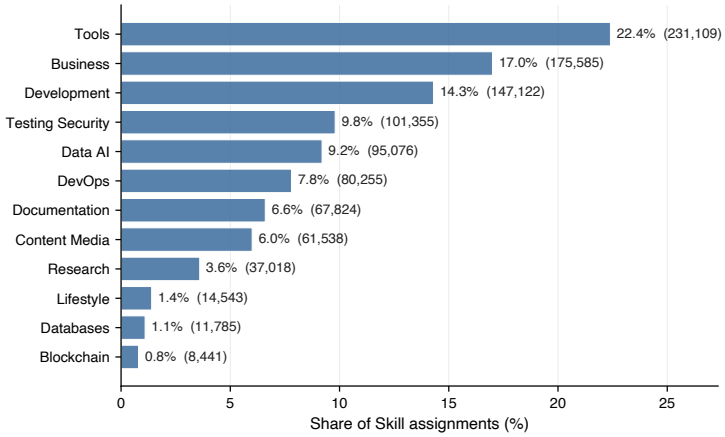

Figure 8: Distribution of Skill categories across 1,031,651 marketplace assignments (snapshot 2026-04-30). No single category exceeds a quarter of the corpus, reflecting broad developer interest in software-engineering tooling, business workflows, testing/security, and data/AI.

Structural Patterns. Most Skills with reachable cloned source are minimal (Figure 9, panel a): 59.9% contain a single file (just `SKILL.md`) and 86.4% contain five or fewer files (n=767,430, median 1, mean 5.6 inflated by the long tail). The file-extension mix (panel b) confirms the ecosystem is documentation-heavy: `.md` alone accounts for 51.7% of all 4.27M files, followed by `.py` (6.9%), `.ts` (6.1%), `.js` (5.7%), and `.json` (4.8%). Natural-language instructions dominate over executable implementations.

A.3 Quality Indicators

We developed a quality scoring rubric based on:

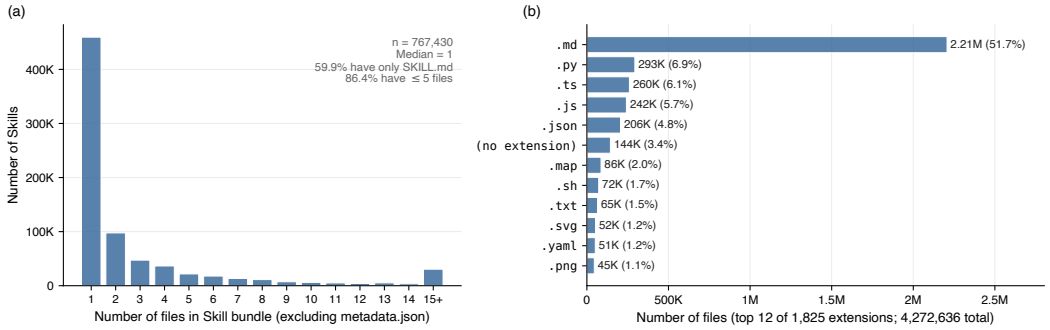

Figure 9: Structural patterns of Skill bundles in the reachable-clone sample. **(a)** File count per bundle (n=767,430, excluding `metadata.json`); most Skills are minimal, with 86.4% containing five files or fewer. **(b)** Top 12 file extensions across the 4.27M files in the sample (out of 1,825 distinct extensions); markdown dominates at 51.7%, with Python, TypeScript, JavaScript, and JSON as the next most common.

1. **Completeness:** Presence of required components (0–3 points)
2. **Clarity:** Readability and organization (0–3 points)
3. **Specificity:** Actionable vs. vague guidance (0–3 points)
4. **Examples:** Presence and quality of examples (0–3 points)

Mean quality score across the ecosystem is 6.2/12 (SD=2.8), indicating substantial room for improvement in Skill authoring practices.

A.4 Implications for Benchmark Design

This ecosystem analysis directly informed SKILLSBENCH construction:

- **Domain selection:** Task categories mirror ecosystem coverage, ensuring Skills exist for evaluation
- **Quality awareness:** Ecosystem mean quality of 6.2/12 motivated our leakage audit and authoring guidelines—low-quality Skills would confound efficacy measurement
- **Skill selection:** We selected benchmark Skills from the top quality quartile (score $\geq$ 9/12) to isolate the effect of procedural knowledge from Skill quality variance
- **Size constraints:** Median Skill size ($\sim$1.8k tokens) informed our 8K context budget allocation

Limitation: Benchmark vs. Ecosystem Gap. Our 87 tasks with high-quality Skills represent an optimistic scenario. Real-world Skill usage involves lower-quality Skills (ecosystem mean: 6.2/12 vs. benchmark mean: 10.1/12) and imperfect Skill-task matching. Future work should evaluate with ecosystem-representative Skill samples.

B Task Specification and Review Process

This appendix provides full details of the task specification format and quality control process summarized in §3. Every task is packaged according to the `task.md` standard—a single schema-validated document plus the role-named `verifier/` and `oracle/` directories—whose full specification is given in Appendix C; this section documents the layout as instantiated by SKILLSBENCH and the review process that gates admission to the benchmark.

B.1 Task Directory Structure

Each task is a self-contained directory with the following layout:

```
tasks/<task-id>/
  task.md                 # YAML frontmatter + task instruction
  environment/
```

```
    Dockerfile            # Container setup
    skills/               # Curated Skills (absent in no-Skills)
      <skill-name>/
        SKILL.md          # Required per skill
        scripts/          # Optional executable code
        references/       # Optional reference documentation
  oracle/
    solve.sh              # Oracle solution (must pass 100%)
  verifier/
    test.sh               # Runs pytest inside the container
    test_outputs.py       # Programmatic assertions
```

B.2 Task Configuration (`task.md` Frontmatter)

Each task specifies metadata and resource limits in the YAML frontmatter of `task.md`; the task instruction is the document body (Appendix C):

```
---
schema_version: "1.3"
metadata:
  author_name: Contributor Name
  author_email: email@example.com
  difficulty: medium      # easy | medium | hard
  category: finance-economics
  tags: [pandas, data-analysis, spreadsheet]
verifier:
  type: test-script
agent: {}
environment:
  network_mode: no-network
  cpus: 1                 # 1-4 cores
  memory_mb: 4096         # 2048-10240 MB
  storage_mb: 10240       # 10 GB standard
---
```

B.3 Verification Infrastructure

Verification typically uses pytest with the CTRF (Common Test Report Format) output (85 of 87 tasks; the remaining two drive a Python checker script directly); every task's `test.sh` writes its reward to `/logs/verifier/reward.txt`. The canonical `test.sh` installs dependencies, runs pytest, and writes a binary reward in the minimal pass/fail case:

```
#!/bin/bash
pip3 install --break-system-packages pytest pytest-json-ctrf
mkdir -p /logs/verifier
pytest --ctrf /logs/verifier/ctrf.json \
       /verifier/test_outputs.py -rA -v
if [ $? -eq 0 ]; then
  echo 1 > /logs/verifier/reward.txt
else
  echo 0 > /logs/verifier/reward.txt
fi
exit 0
```

A task passes if and only if all assertions in `test_outputs.py` succeed (reward = 1); otherwise the verifier records a fail (reward = 0).

B.4 PR Review Process

Each submitted task undergoes a multi-stage review:

1. **Automated CI**: Structural validation (`bench tasks check`), oracle execution (`bench eval create -agent oracle`, must pass 100%), and AI-detection screening (GPTZero) on the `task.md` instruction body.

2. **Maintainer review**: Evaluates data validity, task realism, oracle quality, Skill quality, and anti-cheating robustness. Reviewers run benchmark experiments with and without Skills across multiple agents.
3. **Benchmark report**: For each task, reviewers produce a structured report documenting oracle results, agent pass rates with and without Skills, failure analysis, and a final verdict (approve, major changes needed, or reject).

Of 400 candidate submissions from 142 contributors, 87 tasks passed all review stages and are included in the current benchmark inventory (22% acceptance rate). The empirical study reports near-complete paired public trajectories for the 87-task evaluation.

B.5 Contributor Checklist

All task submissions must satisfy the following requirements before review:

- ☐ The `task.md` instruction is human-written (not model-generated); verified via GPTZero and human review
- ☐ Skills are error-free, factually correct, and generalizable to similar tasks
- ☐ Task is realistic and grounded in professional workflows
- ☐ Task requires domain knowledge and is significantly easier with Skills
- ☐ Outputs are deterministic and verifiable with programmatic assertions
- ☐ Test count is below 10 unless justified; tests cover distinct criteria
- ☐ Oracle solution passes 100% (`bench eval create -agent oracle`)
- ☐ Task tested with agent both with and without Skills

B.6 Maintainer Review Policy

Maintainers evaluate each submission against seven criteria:

1. **AI detection**: Verify the `task.md` instruction and configuration are manually written using GPTZero and human review. PRs with intentional grammar errors designed to circumvent AI detectors are closed.
2. **Data quality**: Data must be real-world and appropriately complex. AI-generated or toy data is rejected.
3. **Task validity**: Tasks must be grounded in realistic professional scenarios. Artificially inflated complexity is rejected.
4. **Oracle quality**: Simple solutions (e.g., an Excel formula or short script) are preferred over over-engineered oracle implementations.
5. **Author history**: Authors flagged multiple times across PRs are closed automatically.
6. **Test parsimony**: Fewer than 10 test cases unless justified; tests should cover distinct criteria rather than repeat similar checks.
7. **Multimodal verification**: For multimodal tasks (audio, PPTX, video, PDF), maintainers personally inspect agent output to verify correctness beyond programmatic assertions.

B.7 Automated CI Pipeline

The CI pipeline performs the following checks on each PR:

- **Structural validation** (`bench tasks check`): Verifies required files exist, the `task.md` frontmatter schema is valid, Dockerfile builds, and test structure is correct.
- **Oracle execution** (`bench eval create -agent oracle`): Runs the oracle solution end-to-end and requires 100% test pass rate.
- **AI-detection screening**: Runs GPTZero on the `task.md` instruction body to flag potential model-generated content.
- **LLM-backed quality checks**: Automated verification of behavior consistency between instructions and tests, anti-cheating measures, pinned dependency checks, typo detection, and hardcoded solution detection.

B.8 Benchmark Report Template

For each task, reviewers produce a structured report documenting:

1. **Task metadata**: Name, category, difficulty, tags, description, Skills provided, key requirements.
2. **Oracle results**: Pass/fail status, reward, tests passed, timing.
3. **Agent results**: Pass rates per agent-model combination, with and without Skills, including execution time.
4. **Skills impact**: Quantified comparison of with-Skills vs. without-Skills performance per agent.
5. **Failure analysis**: Per-test breakdown of failures including actual vs. expected output, root cause, and evidence from trajectories.
6. **Recommendation**: One of: APPROVE, APPROVE WITH CAVEATS, MAJOR CHANGES NEEDED, or REJECT.

B.9 Review Lifecycle

PRs progress through a defined label-based lifecycle:

1. **WIP** → **Need review**: Author signals readiness for initial review.
2. **Need review** → **Reviewing**: A maintainer begins active review and benchmark experiments.
3. **Reviewing** → **Change requested / Major change needed / Critical change needed**: Issues identified; author must address.
4. **Change requested** → **Take another look**: Author responds after changes.
5. **Ready to merge** → **Good task**: All reviews passed; task included in benchmark.

Critical changes include unrealistic task scenarios, AI-generated instructions, or synthetic data. Major changes include incorrect tests, unreliable verifiers, or poor Skill quality requiring re-evaluation.

B.10 Task Quality Criteria

Tasks are evaluated against the following criteria:

- **Realistic**: Grounded in professional workflows that people in that domain actually perform
- **Skill-dependent**: Significantly easier with Skills than without—tasks solvable without any procedural guidance are rejected
- **Verifiable**: Deterministic outputs testable with programmatic assertions; LLM-as-judge is not used
- **Composable**: Tasks should exercise 3–6 Skills together; instructions never reference which Skills to use
- **Test parsimony**: Fewer than 10 test cases unless justified; tests should cover distinct criteria rather than repeat similar checks

C The `task.md` Task Standard

We claim in §1 that the paired-evaluation protocol is reusable beyond our own task set: practitioners can run the same harness on their own Skill libraries before shipping. That reuse requires the task package to be portable and versioned. The `task.md` standard therefore packages a task as a single schema-validated document—YAML frontmatter for configuration, Markdown body for the task instruction—together with runtime directories named for their roles (`verifier/`, `oracle/`), so that all 87 evaluated tasks load through one parser. Appendix B shows the package layout as instantiated by SKILLSBENCH; this appendix specifies the standard itself.

Every pass rate reported in this paper comes from the deterministic `test-script` verifier described below. The standard is strictly more expressive than SKILLSBENCH requires: it defines verifier strategies the benchmark deliberately does not use (Table 4), so the format's generality does not come at the expense of the no-LLM-as-a-judge determinism on which the main measurement depends (§3). The enforcement properties stated below hold in the open-sourced SKILLSBENCH reference harness [BenchFlow team, 2026] and are covered by its test suite. Beyond the reference harness, the standard is supported by the AgentBeats platform: tasks packaged as `task.md` run directly on AgentBeats, where SKILLSBENCH is publicly available.[1]

[1] `https://agentbeats.dev/Yiminnn/skillsbench-agentbeats`

Although initially based on the Harbor task format [Harbor Framework Team, 2026], SKILLSBENCH tasks required a new approach to support the self-generated Skills condition (Appendix D.6). This condition relies on two isolated agent sessions (a creator and a clean solver), whereas Harbor only supports single, continuous rollouts. To resolve this, we developed the `task.md` standard. This format extends Harbor's containerized framework by merging configuration and instruction files into a single schema-validated document, adding controlled metadata and explicit verifier strategies (Table 4), and renaming the `solution/` and `tests/` directories to `oracle/` and `verifier/` to better reflect their roles.

C.1 Document Structure

`task.md` is a Markdown file whose YAML frontmatter is the task configuration and whose body, immediately following the closing frontmatter delimiter, is the task instruction—no section heading is required. The schema is strict at the top level: unknown keys are rejected at parse time, so a contributor's task cannot silently lose a configuration field as the schema evolves across harnesses; values *within* the free-form `metadata` block are open by design (next subsection).

```
---
schema_version: "1.3"
metadata:
  difficulty: medium           # easy | medium | hard
  category: finance-economics  # controlled vocabulary
  task_type: [analysis]
  tags: [pandas, spreadsheet]  # open tier
environment:
  network_mode: no-network     # no-network|public|allowlist
  cpus: 1
  memory_mb: 4096
  storage_mb: 10240
agent: {}
verifier: {type: test-script}
oracle: {}
---
You are given ... produce ... in /app/out.json.
```

C.2 Controlled Metadata Vocabulary

The `metadata` block is two-tier. A *closed* tier—`category`, `task_type`, `modality`, `interface`, `skill_type`, and `difficulty`—draws from controlled vocabularies, of which `category` and `difficulty` correspond to the benchmark taxonomy of Appendix M.9; an *open* tier (`tags` and free-form Skill names) preserves authoring flexibility. The closed tier is enforced at review time rather than at parse time: `metadata` is parsed as a free-form mapping with no load-time enumeration check, so off-taxonomy values are surfaced during review rather than rejected by the parser. This keeps authoring lightweight while still yielding the benchmark-wide classification over which the per-domain breakdowns are computed.

C.3 Explicit Network Policy

Because uncontrolled network egress is a confound for a deterministic verifier, the standard makes egress explicit: access is declared with a single enumerated field, `network_mode` $\in$ {`no-network`, `public`, `allowlist`}, rather than a coarse on/off boolean. The `allowlist` mode requires a non-empty `allowed_hosts` list—a parse-time error otherwise—so an `allowlist` declaration is never left under-specified. A task whose data is baked into the container image declares `no-network`; a task that must reach a fixed external service declares `allowlist` with the specific hosts.

C.4 Verifier Strategies

A verifier strategy is selected by a `type` field, either in the `verifier` block of `task.md` or in an optional `verifier/verifier.md` manifest, where the same strategy is spelled `script` rather than `test-script` (Table 4). Absent a manifest, the configuration's `verifier.type` selects the default `test-script` path, which runs `verifier/test.sh` and reads a pass/fail result from `reward.txt`

(or a structured `reward.json`); `test-script` is the only strategy the 87 tasks use, which keeps the measurement free of LLM-as-a-judge variance (§3). The remaining strategies—`llm-judge`, `reward-kit`, and `agent-judge` / `ors-episode`—exist for task classes whose correctness a deterministic script cannot decide; they are part of the standard but not of this benchmark. Unscored, stale, or rate-limited rows are treated as incomplete coverage; timeout rows enter only when healthy replacements are unavailable and are scored as failures.

Table 4: Verifier strategies defined by the `task.md` standard. SKILLSBENCH uses `test-script` exclusively to preserve deterministic, programmatic verification.

Strategy	Reward source	In SKILLSBENCH
`test-script`	`test.sh` → `reward.txt/json`	all 87 tasks
`llm-judge`	rubric-scored deliverables	not used
`reward-kit`	external reward entrypoint	not used
`agent-judge`	judged transcript	not used
`ors-episode`	episode evidence → reward	not used

D Experimental Setup Details

This appendix expands the experimental setup summarized in §4.

D.1 Setup overview

We instantiate the SKILLSBENCH protocol across commercial terminal agents and model–harness configurations. The latest aggregate targets a fixed 9,396-slot frame (18 configurations $\times$ 87 tasks $\times$ 2 conditions $\times$ 3 trials), with 9,396 selected public result files currently available. All runs use temperature 0; unscored, stale, or rate-limited rows are tracked as coverage gaps for audit and rerun, while timeout rows are used only when healthy pass/fail replacements are unavailable and are scored as failures.

Models. The latest aggregate pairs OpenHands with GPT-5.5, GPT-5.4 Mini, Claude Opus 4.8/4.7, Claude Sonnet 4.6, Gemini 3.1 Pro, Gemini 3.5 Flash, Gemini 3.1 Flash Lite, GLM 5.1, Kimi K2.6, DeepSeek V4 Pro/Flash, Grok 4.3, and MiniMax M3/M2.7; Gemini CLI with Gemini 3.1 Pro; Claude Code with Claude Opus 4.7; and Codex CLI with GPT-5.5. Full identifiers and selected result counts are listed in Table 5; protocol-level details appear in Appendix N.

Metrics. Our primary metric is task-macro pass rate following Terminal-Bench [Merrill et al., 2026]: verifier-scored pass/fail outcomes are averaged over repeated trials within each task and then averaged across the fixed inventory of 87 tasks. We report absolute Skills improvement and normalized gain $g = (\text{pass}_{\text{skill}} - \text{pass}_{\text{vanilla}})/(1 - \text{pass}_{\text{vanilla}})$ as defined in the main text (§4). We interpret g alongside absolute pass-rate deltas to avoid ceiling-effect artifacts. Harness versions, resource limits, retry rules, and confidence-interval calculations are provided in Appendix N.

D.2 Model and Harness Configurations

Table 5 presents the 18 model–harness configurations evaluated in the latest aggregate.

D.3 Harness Descriptions

We evaluate four commercial agent harnesses:

- **OpenHands**: open-source terminal agent harness
- **Claude Code** [Anthropic, 2025b]: Anthropic's agent with native Skill integration
- **Gemini CLI** [Google, 2025]: Google's open-source terminal agent
- **Codex CLI** [OpenAI, 2025]: OpenAI's lightweight coding agent

Table 5: Agent harnesses and models evaluated in the latest aggregate. Counts report selected no-Skills / curated-Skills public result files out of 261 possible per condition.

Harness	**Model**	**Provider**	n
OpenHands	GPT-5.5	OpenAI / Azure OpenAI	261/261
Codex	GPT-5.5	OpenAI / Azure OpenAI	261/261
Claude Code	Opus 4.7	Anthropic	261/261
Gemini CLI	Gemini 3.1 Pro	Google	261/261
OpenHands	GLM 5.1	Z.ai / GLM	261/261
OpenHands	Claude Opus 4.8	Anthropic / AWS Bedrock	261/261
OpenHands	Kimi K2.6	Moonshot AI	261/261
OpenHands	Claude Opus 4.7	Anthropic / AWS Bedrock	261/261
OpenHands	MiniMax M3	MiniMax	261/261
OpenHands	Gemini 3.1 Pro	Google	261/261
OpenHands	DeepSeek V4 Pro	DeepSeek	261/261
OpenHands	Gemini 3.5 Flash	Google	261/261
OpenHands	Claude Sonnet 4.6	Anthropic / AWS Bedrock	261/261
OpenHands	DeepSeek V4 Flash	DeepSeek	261/261
OpenHands	Grok 4.3	xAI	261/261
OpenHands	GPT-5.4 Mini	OpenAI	261/261
OpenHands	MiniMax M2.7	MiniMax	261/261
OpenHands	Gemini 3.1 Flash Lite	Google	261/261

These tightly couple specific models with proprietary agent logic, representing real-world deployment conditions.

Model Family Consideration. Claude models have been trained with awareness of the Agent Skills specification [Anthropic, 2025a], which may confer advantages when processing Skill-formatted instructions.

D.4 Agent Interface

Agents interact with the environment through a standardized interface:

```
class BaseAgent(ABC):
    @abstractmethod
    def step(self, obs: str) -> str:
        """obs: terminal output -> action"""
        pass
```

D.5 Skill Structure

Each Skill is a directory containing a required `SKILL.md` file with YAML frontmatter and optional bundled resources:

```
skill-name/
  SKILL.md          # Required: YAML frontmatter + instructions
  scripts/          # Optional: executable code
  references/       # Optional: reference documentation
```

The `SKILL.md` frontmatter specifies the Skill's name and a one-line description used by agents for skill discovery. The body contains procedural guidance, code examples, and usage patterns.

D.6 Self-Generated Skills Condition

The self-generated condition asks whether an agent can replace curated Skills with Skills it authors for itself. Unlike the no-Skills and curated-Skills conditions, it consists of *two isolated sessions* per

task. In the *creator* session, a clean agent receives the task package with all curated Skills removed and exactly one Skill mounted—Anthropic's official `skill-creator`—and is instructed to author skill packs rather than solve the task. The creator prompt, verbatim from the harness:[2]

```
Use the skill-creator skill exactly as provided.

Read /instruction.md and inspect the task environment only as
needed to understand the reusable workflow.  Do not solve the task
directly.

Create one or more complete Anthropic-standard skill packs as
immediate child directories under:  /app/generated-skills

Use this suggested path if one skill is enough:
/app/generated-skills/<task>-skill

Each generated skill pack path must look like
/app/generated-skills/<skill-name>/SKILL.md.  It may include
scripts/, references/, assets/, examples, or other bundled resources
when they help a fresh solver avoid repeated work.

The solver context will start with a clean agent session and only
the generated skill packs mounted.  Make the skills useful for
solving this task type from the same sandbox environment.
```

In the *solver* session, a fresh agent then runs the task exactly as in the curated-Skills condition, except that the curated Skills are replaced by the generated packs; the task instruction itself is unmodified, and none of the creator's reasoning is in the solver's context. The campaign realized this protocol in two ways: the Claude Code runs execute the creator and solver scenes back-to-back inside each trial's sandbox, regenerating the packs every trial, whereas the Codex and Gemini CLI runs generated the packs once per (task, configuration) and reused them across the scored solver trials. Because the Claude Code scenes share the task sandbox, file-system state can carry over beyond the pack files; the consequences of both realizations are examined in Appendix D.6.1.

Configurations and accounting. The condition is evaluated on the three dedicated-harness configurations of the main aggregate—Claude Code with Opus 4.7, Codex with GPT-5.5, and Gemini CLI with Gemini 3.1 Pro—using the same harness, model identifier, and sandbox as their baselines.[3] The self-generated condition is retained as a diagnostic rather than part of the main two-condition aggregate. The current audit reports verifier-scored trials against the same 87-task, three-trial frame: 258, 258, and 253 of the 261 slots are present for the three configurations respectively (the campaign skipped `pddl-airport-planning`); unscored or missing slots are treated as coverage gaps rather than accepted outcomes. The no-Skills and curated-Skills columns restate the main aggregate of Table 2.

Table 6: Self-generated Skills vs. the matched baselines on the three dedicated-harness configurations (pass rate %, fixed 87-task denominator). Δ_G = self-generated minus no Skills; Δ_S = curated Skills minus no Skills.

Harness	Model	No Sk.	Self-Gen	Δ_G	Curated	Δ_S
Claude Code	Opus 4.7	43.0	34.9	–8.1	61.2	+18.2
Codex	GPT-5.5	46.8	35.5	–11.3	66.5	+19.7
Gemini CLI	Gemini 3.1 Pro	36.0	24.5	–11.5	60.8	+24.8

On all three configurations, self-generated Skills land *below* the no-Skills baseline (-8.1 to -11.5 pp) while curated Skills add $+18.2$ to $+24.8$ pp (Table 6, Figure 10). The trajectory audit below shows the deficit is not a single mechanism: generated packs frequently go unused by the solver, creator-side authoring can displace solver progress, and packs that are used can lock in confidently wrong assumptions.

[2] `/instruction.md` is the in-sandbox mount of the task instruction (the `task.md` body); `<task>` denotes the task name.

[3] One deviation: the Claude Code self-generated runs used effort level `max`, whereas its baseline runs used `high`; the Codex (`xhigh`) and Gemini CLI configurations match their baselines exactly.

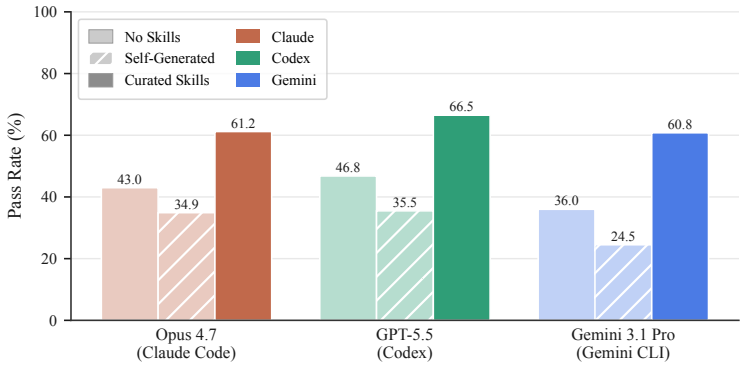

Figure 10: Self-generated Skills underperform not only curated Skills but the no-Skills baseline on all three dedicated-harness configurations.

D.6.1 Why self-generated Skills underperform: trajectory audit

We audited twelve solver trajectories across ten (task, configuration) pairs selected for extreme condition gaps—tasks where curated Skills pass but self-generated fail, the rare cases where self-generated beats no-Skills, and skill-dependent tasks—together with a protocol check comparing trials of the same task. Four mechanisms account for the audited gap:

- **Generated packs frequently go unused.** In the pre-baked Codex and Gemini CLI family, 10 of 12 audited trajectories never list, read, or mention the generated skills—the event stream contains zero occurrences of "skill"—so those passes *and* failures are causally unrelated to pack content. Unlike curated Skills, which the harnesses surface through their native discovery mechanisms, a pack the solver never discovers can only be dead weight.
- **Authoring displaces task work.** In the Claude Code realization the creator and solver scenes share a single trial context. On `threejs-to-obj`, creator-side work consumed the trajectory before a solver artifact was produced—trial logs end with the `SKILL.md` write still pending—even though the creator had already produced and verified a correct OBJ export (written only to `/tmp`, obeying "Do not solve the task directly"). Both baselines pass this task; the self-generated miss is creator/solver interference, not knowledge failure.
- **Consumed packs lock in confident errors.** When the solver does use a generated pack, wrong assumptions become load-bearing. On `3d-scan-calc`, the generated `SKILL.md` asserts as a "critical gotcha" that the STL coordinates are millimetres and hard-codes a $1000\times$ unit conversion—both solver trials ran the bundled script unquestioned and submitted a mass off by three orders of magnitude, with every other step correct. The curated Skill carries exactly the opposite warning ("Do not assume millimeters . . . no unit conversion is needed"). On `radar-vital-signs` the generated pack replaced the curated multi-check pipeline with a fire-and-forget script whose plausibility bounds were too loose to flag its own implausible output, omitting the curated autocorrelation cross-check.
- **The clearest win is leakage, not reuse.** The strongest positive case, `fix-visual-stability` (self-generated 1.0 vs. curated 0.0 on Claude Code), works because the creator wrote the pack *inside the graded sandbox*: the generated `SKILL.md` names the app's exact offender components, enumerates every `data-testid` the verifier checks, and prescribes a fix order—an answer key for this instance rather than reusable procedure. The curated Skill ships only generic guidance plus measurement tooling.

Two caveats temper per-task readings (the aggregate direction in Table 6 is unaffected). First, some zeros are measurement artifacts: the `3d-scan-calc` verifier was patched ten days after these runs to also accept the millimetre interpretation, and unscored audit records are excluded from causal interpretation. Second, the audit's visibility is bounded: creator sessions for the pre-baked family are not part of the released artifacts, and Gemini CLI trajectories omit tool outputs, so non-discovery there is established behaviorally rather than from pack content.

D.7 Software and Model Versions

Table 7 lists representative model identifiers and harnesses used in the latest aggregate.

Table 7: Representative model API identifiers and harnesses in the latest aggregate.

Display Name	API Model ID	Harness Version
GPT-5.5	`openai/gpt-5.5`	OpenHands / Codex CLI
GPT-5.4 Mini	`openai/gpt-5.4-mini`	OpenHands
Claude Opus 4.8	`claude-opus-4-8`	OpenHands
Claude Opus 4.7	`claude-opus-4-7`	OpenHands / Claude Code
Claude Sonnet 4.6	`claude-sonnet-4-6`	OpenHands
Gemini 3.1 Pro	`gemini-3.1-pro-preview`	OpenHands / Gemini CLI
Gemini 3.5 Flash	`gemini-3.5-flash`	OpenHands
Gemini 3.1 Flash Lite	`gemini-3.1-flash-lite-preview`	OpenHands
GLM 5.1	`glm/glm-5.1`	OpenHands
Kimi K2.6	`kimi/kimi-k2.6`	OpenHands
Grok 4.3	`xai/grok-4.3`	OpenHands
DeepSeek V4 Pro	`deepseek/deepseek-v4-pro`	OpenHands
DeepSeek V4 Flash	`deepseek/deepseek-v4-flash`	OpenHands
MiniMax M3	`minimax/MiniMax-M3`	OpenHands
MiniMax M2.7	`minimax/MiniMax-M2.7`	OpenHands

D.8 Container Environment

All tasks run in Docker containers built from an `ubuntu:24.04` base image. Per-task resource allocation is specified in the `environment` block of the task configuration (Appendix C):

- **CPUs**: 1–4 cores (task-dependent)
- **Memory**: 2–10 GB (task-dependent)
- **Storage**: 10 GB (standard across all tasks)
- **GPU**: None (no tasks require GPU)

Containers are deleted after each trial (`delete: true`) to ensure no state leaks between runs.

D.9 Inference Configuration

- **Temperature**: 0 (deterministic sampling)
- **Reasoning effort**: Highest available setting for each model
- **Context management**: Sliding window with 8K token limit; oldest turns dropped when exceeded
- **Run selection**: Public result files with complete metadata and healthy verifier-scored pass/fail outcomes are preferred; timeout rows are used only as failure backfill when healthy replacements are unavailable

D.10 Experiment Orchestration

- **Runs per task**: three selected public result files for the main no-Skills and curated-Skills conditions
- **Concurrent trials**: 256 for the main conditions
- **Retry policy**: Unscored, stale, or rate-limited runs are audited and rerun
- **Task scope**: 87 tasks in the current released inventory

The latest aggregate tracks 9,396 fixed result slots across 18 configurations, 87 tasks, and 2 conditions; 9,396 public result files are currently selected.

E Task-Level Results

Figure 11, Figure 12, and Figure 13 show the task-level pass rates per model across all 87 tasks. Results use the same fixed three-trial denominator as Table 2. Tasks (rows) are sorted by average with-Skills pass rate; models (columns) are sorted by aggregate with-Skills score.

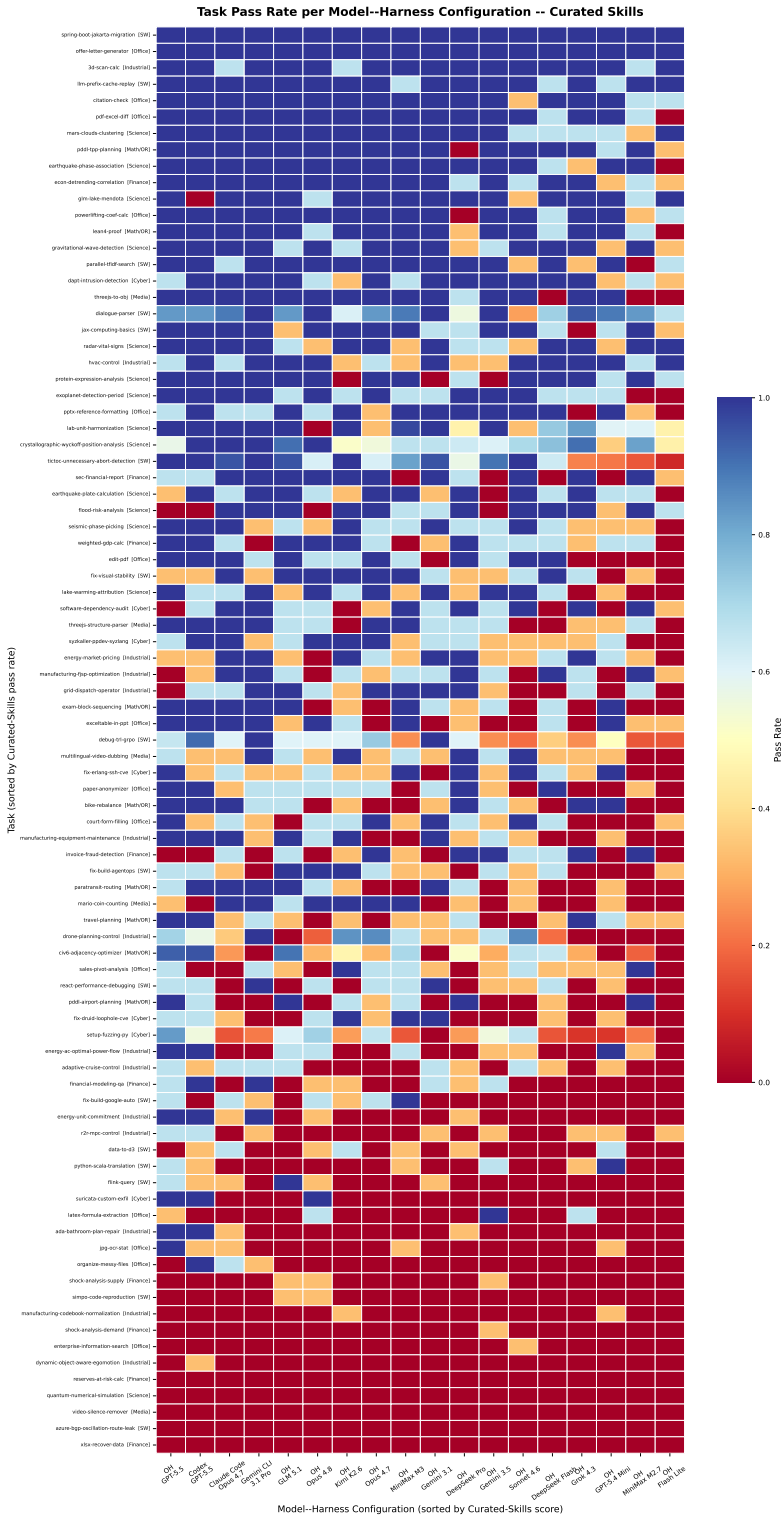

Figure 11: Task pass rate per model with curated Skills. The grid reveals a common set of easy tasks (top, uniformly blue) solved by all models, and hard tasks (bottom, uniformly red) unsolved even with Skills. Tasks along the diagonal transition from solvable to unsolvable as model capability decreases.

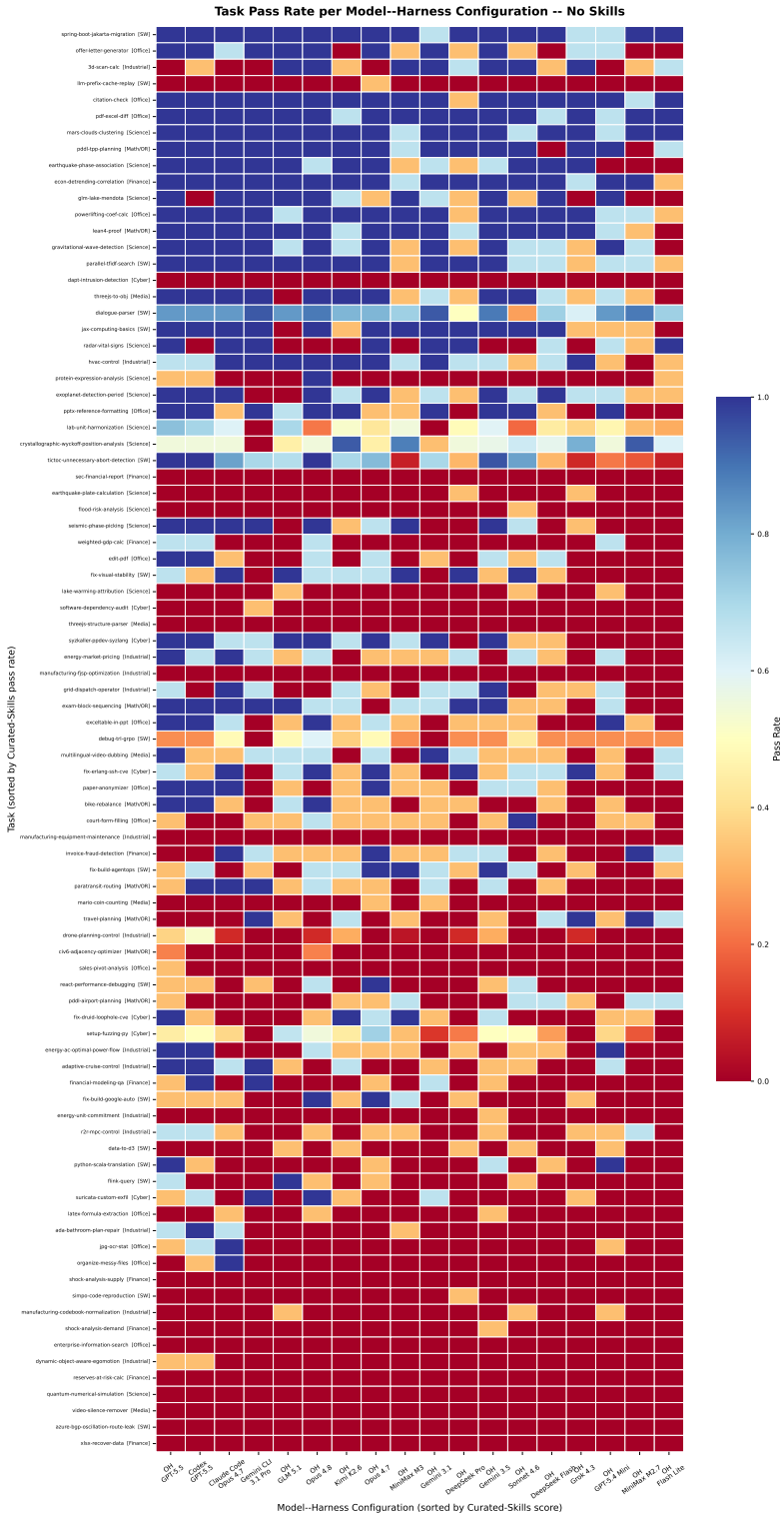

Figure 12: Task pass rate per model without Skills (baseline). Compared to Figure 11, the blue region contracts substantially, confirming that Skills shift many tasks from unsolved to solved.

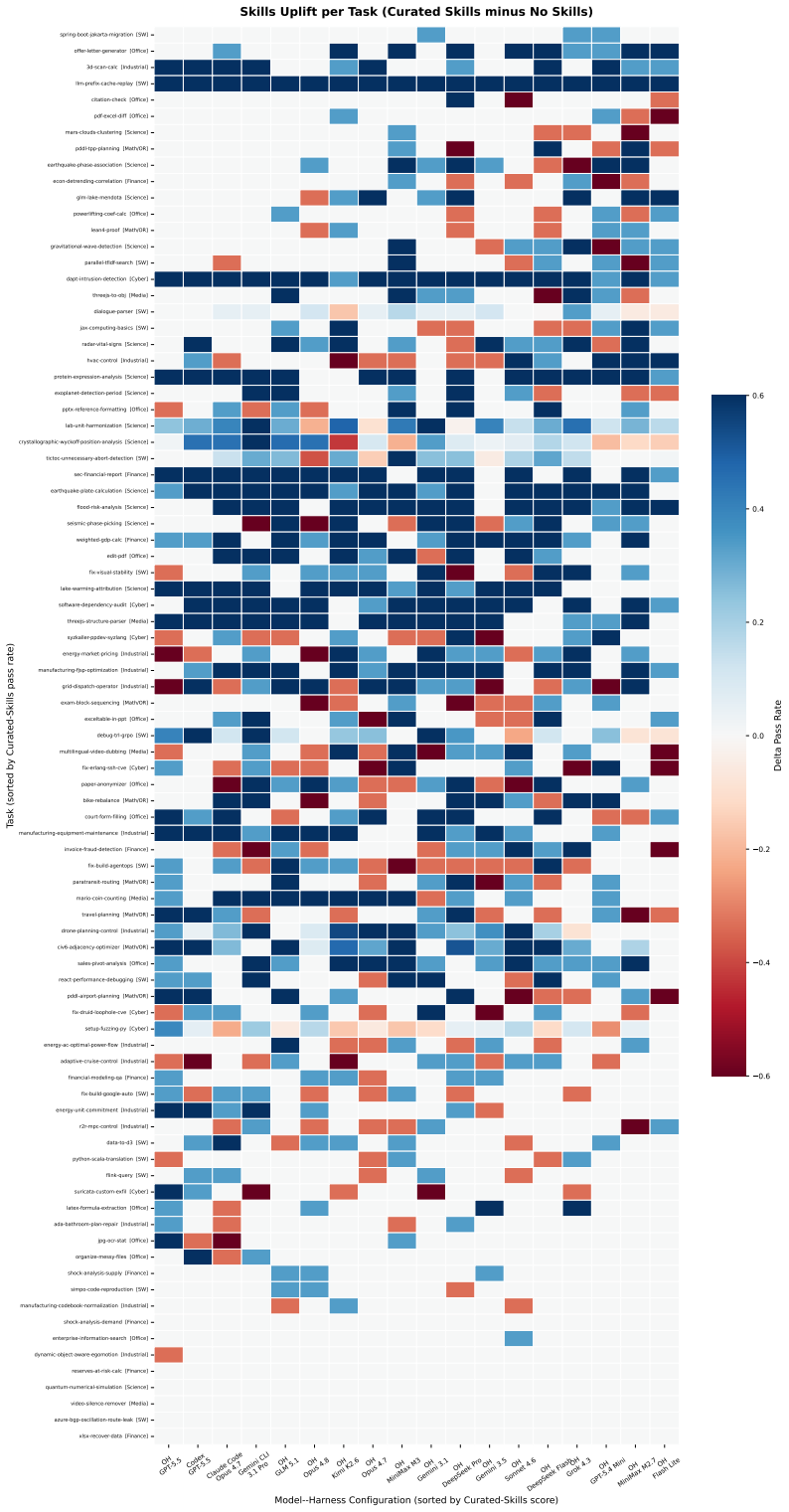

Figure 13: Skills uplift per task (with Skills − without Skills). Blue cells indicate positive uplift; red cells indicate tasks where Skills hurt performance. The majority of cells are blue, confirming broad Skill benefit. A small number of tasks show negative delta for specific models.

F Skill Design Factors

We expand on the design ablations summarized in Finding 5 (§5). All numbers use the latest 87-task, 18-configuration aggregate with the same fixed three-trial denominator as Table 2.

F.1 Skill Quantity

Table 8: Pass rates by number of Skills attached to a task. One to three Skills remain strongest; ≥ 4 Skills introduces diminishing returns consistent with cognitive overhead and conflicting guidance.

Skills Count	N tasks	No Skills	With Skills	Δ_{abs}
1	**23**	**32.2%**	**50.2%**	**+18.0**
2–3	43	34.4%	53.4%	+19.0
≥ 4	21	34.8%	44.9%	+10.1

F.2 Skill Complexity

Table 9: Pass rates by Skill documentation length. Buckets use total `SKILL.md` text size with 25th, 50th, and 95th percentile cutpoints. Focused documentation outperforms comprehensive prose.

Complexity	N tasks	No Skills	With Skills	Δ_{abs}
Compact	22	37.6%	56.6%	+19.0
Standard	22	19.8%	41.2%	+21.5
Detailed	38	39.3%	53.8%	+14.5
Comprehensive	5	39.3%	40.0%	+0.7

F.3 Failure Modes when Skills Hurt

We audited paired (with-Skills, no-Skills) trajectories on tasks with negative deltas. The latest aggregate changes the task ordering, but the same three repeatable patterns remain:

- **Pattern A: heavyweight pipeline crowds out simpler execution.** *Tasks:* `adaptive-cruise-control` (-5.6 pp), `r2r-mpc-control` (-5.6 pp), `dynamic-object-aware-egomotion` (-1.9 pp). The Skill points agents toward more principled but heavier optimization workflows, which can crowd out a simpler path to a valid answer. *Implication:* authors should mark optional steps and supply a fast path.
- **Pattern B: Skill activation displaces a stronger native strategy.** *Tasks:* `exam-block-sequencing` (-7.4 pp), `econ-detrending-correlation` (-5.6 pp), `python-scala-translation` (-1.9 pp). A generic recipe can suppress a stronger direct coding or debugging strategy. *Implication:* Skills should include applicability boundaries, not just preferred procedures.
- **Pattern C: Skill points the agent at a solver it can't debug.** *Tasks:* `suricata-custom-exfil` (-7.4 pp), `mars-clouds-clustering` (-5.6 pp), `fix-erlang-ssh-cve` (-3.7 pp). When a Skill mandates a brittle framework or schema, agents inherit a steeper debug surface and can fail before the model reaches a correct artifact. *Implication:* Skills should provide debugging fallbacks and validation checks.

F.4 Skill-Author Patterns Audit

We audited paired trajectories (with-Skills vs. no-Skills) on the 10 highest-Δ tasks (Top-10 in §5, mean $+67.0$ pp) to extract concrete authoring patterns. The five patterns summarized in §6's "Implications for Skill authoring" paragraph are: (1) ship an executable script with calibrated defaults rather than only describing the algorithm (`llm-prefix-cache-replay`, `sec-financial-report`); (2) name the canonical data

source and parsing quirk (`flood-risk-analysis`, `dapt-intrusion-detection`); (3) encode the exact file-format constraint the verifier inspects (`threejs-structure-parser`); (4) surface the algorithmic invariant the verifier asserts (`protein-expression-analysis`, `earthquake-plate-calculation`, `manufacturing-fjsp-optimization`); (5) make the Skill's `description:` frontmatter so task-specific that the agent matches it on the first scan. Each pattern is grounded in paired trajectory excerpts, available alongside the released trajectories.

G Confidence Interval Calculation

For the main result bars in Figure 1 and the Skill Invocation Rate plot in Figure 5, we compute 95% binomial (Wald) confidence intervals as $1.96\sqrt{p(1-p)/n}$, where p is the corresponding rate and n is the selected public result-file count for that model–harness cell. Resolution-rate intervals are plotted as slightly offset circles within each stacked bar; invocation intervals are plotted as black diamonds with vertical ranges. Normalized gain is reported as a point estimate.

H Complete Task List

The complete 87-task inventory and updated domain distribution are reported in Table 19. The table is generated from `scripts/taxonomy.csv` and grouped by the updated 8-domain taxonomy.

I Comprehensive Results Summary

Table 10 consolidates the latest aggregate results across 18 model–harness configurations and two matched Skills conditions. Scores are computed on the fixed 87×3 trial frame per condition, consistent with the task-macro scoring used in the main text; the n column reports current public coverage.

Table 10: Comprehensive latest results across model–harness configurations. Pass rates use a fixed 87×3 denominator per condition; Δ is curated-Skills improvement and g is normalized gain. Models are ordered by with-Skills pass rate; n reports selected no-Skills / curated-Skills result files.

Harness	Model	No Sk.	With Sk.	Δ (pp)	g (%)	n
OpenHands	GPT-5.5	51.5	67.3	+15.8	32.6	261/261
Codex	GPT-5.5	46.8	66.5	+19.7	37.0	261/261
Claude Code	Opus 4.7	43.0	61.2	+18.2	31.9	261/261
Gemini CLI	Gemini 3.1 Pro	36.0	60.8	+24.8	38.7	261/261
OpenHands	GLM 5.1	32.7	58.4	+25.7	38.1	261/261
OpenHands	Claude Opus 4.8	45.7	54.1	+8.4	15.5	261/261
OpenHands	Kimi K2.6	33.4	54.0	+20.6	31.0	261/261
OpenHands	Claude Opus 4.7	42.1	53.1	+11.1	19.1	261/261
OpenHands	MiniMax M3	29.7	53.0	+23.3	33.2	261/261
OpenHands	Gemini 3.1 Pro	33.8	52.8	+19.0	28.7	261/261
OpenHands	DeepSeek V4 Pro	26.9	50.1	+23.2	31.8	261/261
OpenHands	Gemini 3.5 Flash	41.1	48.2	+7.1	12.1	261/261
OpenHands	Claude Sonnet 4.6	33.5	47.2	+13.6	20.5	261/261
OpenHands	DeepSeek V4 Flash	27.5	44.7	+17.2	23.7	261/261
OpenHands	Grok 4.3	22.8	41.7	+18.8	24.4	261/261
OpenHands	GPT-5.4 Mini	29.9	41.4	+11.5	16.4	261/261
OpenHands	MiniMax M2.7	18.1	34.9	+16.8	20.5	261/261
OpenHands	Gemini 3.1 Flash Lite	16.0	20.1	+4.1	4.9	261/261
Mean		**33.9**	**50.5**	**+16.6**	**25.5**	**4698/4698**

Key observations.

- Curated Skills improve performance by +16.6 pp on average (range: +4.1 to +25.7 pp), corresponding to a normalized gain of 25.5%.
- OpenHands + GPT-5.5 achieves the highest absolute pass rate (67.3%) with Skills.

- OpenHands + GLM 5.1 shows the largest absolute improvement (+25.7 pp), while Gemini CLI + Gemini 3.1 Pro has the highest normalized gain (38.7%).
- The current public snapshot is complete for all 18 model–harness configurations.

Figure 14 breaks the same aggregate down by model family in the time–performance plane (mean agent wall-clock per task; measurement details in Appendix L.4): every family improves with curated Skills, none at a material time cost.

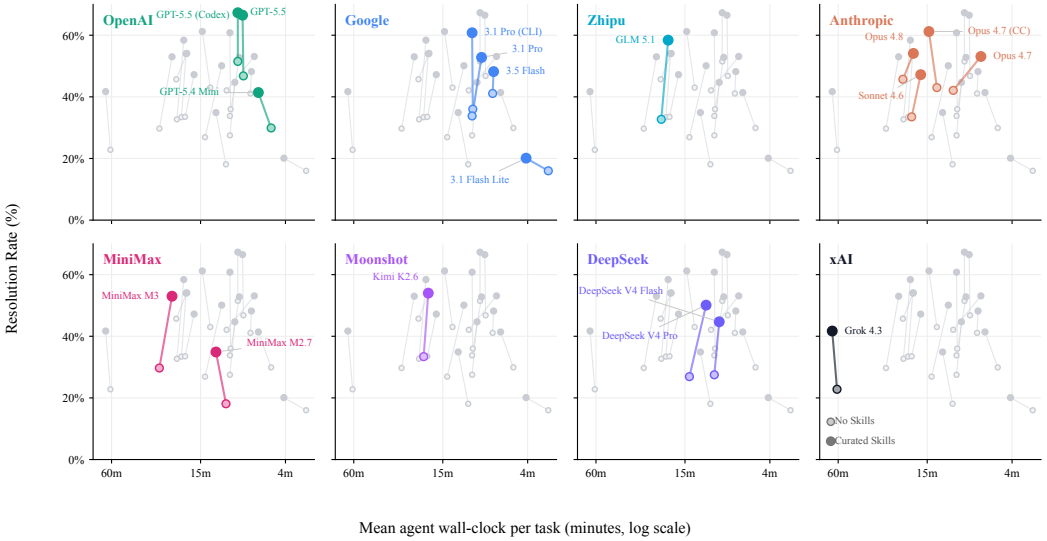

Figure 14: Per-family shift from no Skills (hollow) to curated Skills (solid) in the time–performance plane; gray shows the remaining fleet.

I.1 Auxiliary Failure-Mode Snapshot

Table 11: Failure mode distribution by condition in the latest healthy-first diagnostic selection. Fail Rate uses the fixed 4,698-slot condition denominator; mode shares are percentages of that condition's selected non-solved trials. The selected public coverage spans the full 9,396-slot frame.

Condition	Fail Rate	Timeout	Execution	Coherence	Verification
No Skills	66.1%	1.9%	0.0%	7.9%	90.3%
With Skills	49.5%	3.1%	0.0%	9.1%	87.8%

Table 12: Outcome distribution by configuration (both Skills conditions pooled; % of each configuration's selected trials). Ordered by solved rate.

Harness	Model	Solved	Partial	Attempted	Runtime err.
OpenHands	GPT-5.5	55.2%	6.5%	38.3%	0.0%
Codex	GPT-5.5	52.9%	5.6%	41.6%	0.0%
Claude Code	Opus 4.7	49.0%	5.4%	45.6%	0.0%
Gemini CLI	Gemini 3.1 Pro	48.1%	0.6%	51.3%	0.0%
OpenHands	Claude Opus 4.8	47.3%	4.2%	48.5%	0.0%
OpenHands	Claude Opus 4.7	43.9%	5.7%	50.4%	0.0%
OpenHands	Gemini 3.1 Pro	42.9%	0.8%	56.3%	0.0%
OpenHands	Gemini 3.5 Flash	41.6%	5.6%	52.9%	0.0%
OpenHands	GLM 5.1	41.4%	6.1%	52.5%	0.0%
OpenHands	Kimi K2.6	40.6%	5.6%	51.9%	1.9%
OpenHands	MiniMax M3	38.9%	5.4%	55.7%	0.0%
OpenHands	Claude Sonnet 4.6	38.3%	4.2%	47.7%	9.8%
OpenHands	DeepSeek V4 Pro	35.4%	6.5%	56.7%	1.3%
OpenHands	GPT-5.4 Mini	33.0%	5.7%	61.3%	0.0%
OpenHands	DeepSeek V4 Flash	32.8%	6.7%	60.0%	0.6%
OpenHands	Grok 4.3	30.1%	5.7%	64.2%	0.0%
OpenHands	MiniMax M2.7	23.9%	5.6%	69.2%	1.3%
OpenHands	Gemini 3.1 Flash Lite	15.9%	5.6%	78.5%	0.0%

J Task-Level Case Studies

The following task-level audits use verifier-scored pass/fail outcomes to illustrate how curated Skills change agent behavior. They are qualitative supplements to the aggregate pass-rate tables above.

J.1 Success Case Studies

We present representative examples where curated Skills transformed agent outcomes from failure to success, illustrating the mechanisms through which procedural knowledge improves performance.

Skills bridge domain-specific API gaps: `sales-pivot-analysis`**.** Without Skills, evaluated agents rarely solved this task, which requires creating Excel pivot tables programmatically from population and income data. Agents consistently loaded the data correctly but failed at pivot table creation—Codex attempted manual DataFrame reshaping instead of using `openpyxl`'s pivot table API, producing structurally incorrect output (10/23 tests failed with "list index out of range" on missing pivot objects). With Skills providing step-by-step guidance for the `openpyxl` pivot table workflow, average pass rate rises from 1.9% to 40.7% (+38.9 pp).

Skills provide critical data processing pipelines: `flood-risk-analysis`**.** This task requires identifying flood-risk stations from USGS streamflow data using return period estimation. Without Skills, agents attempted ad-hoc statistical approaches—e.g., simple threshold-based detection or incorrect distribution fitting—achieving only 1.9% pass rate. The curated Skill specified the Log-Pearson Type III distribution, the standard USGS methodology for flood frequency analysis, including the exact scipy function calls and parameter interpretation. With Skills, pass rate rose to 68.5% (+66.7 pp), with many configurations correctly applying the USGS-standard methodology.

Skills encode regulatory knowledge: `sec-financial-report`**.** Analyzing hedge fund activities from SEC 13F filings requires understanding specific regulatory formats, CIK lookup procedures, and filing comparison methodology. Without Skills, no model could complete the task (0% pass rate)—agents either failed to locate the correct filings or misinterpreted the tabular data format. The

curated Skill documented the SEC EDGAR API endpoints, 13F-HR filing structure, and cross-quarter comparison methodology. With Skills, pass rate reached 68.5% (+68.5 pp).

Skills prevent common implementation pitfalls: `manufacturing-fjsp-optimization`**.** The flexible job-shop scheduling problem requires constraint-aware optimization with machine downtime windows. Without Skills, agents produced naive schedules ignoring maintenance constraints (0% pass rate). The curated Skill outlined the constraint propagation approach, objective function formulation, and OR-Tools solver configuration. With Skills, agents successfully formulated and solved the optimization problem in a subset of configurations (55.6% pass rate, +55.6 pp).

J.2 Qualitative Failure Examples

We present representative examples drawn from verifier test outputs across failed trajectories.

Quality Below Threshold: `earthquake-plate-calculation`**.** The agent correctly identified the target earthquake event, extracting latitude, longitude, magnitude, and timestamp accurately (7/8 tests passed). However, the computed distance from the nearest plate boundary was 3,562 km instead of the expected 3,878 km—an 8.2% error that exceeded the ± 0.01 km tolerance. The agent applied the correct Haversine formula but used an incorrect plate boundary coordinate, demonstrating that even when agents understand the computational method, domain-specific data interpretation remains error-prone.

Incomplete Solution: `shock-analysis-supply`**.** The agent created a structurally correct Excel workbook and passed 6 of 9 tests, correctly setting up sheet structures, formula templates, and some data imports. However, it failed to: (1) populate employment/labor data from the Penn World Tables (PWT), (2) execute the HP filter optimization solver, and (3) compute the depreciation rate. These three missing components represent the most domain-specific and computationally demanding aspects of the task.

No Output Produced: `gh-repo-analytics`**.** All 8 verifier tests errored at the fixture stage with "Missing `/app/report.json`," meaning the agent never created the required output file. The task requires interacting with a local Gitea server, cloning repositories, and computing analytics—a multi-step pipeline where failure at any early stage prevents all downstream output.

Specification Violation: `latex-formula-extraction`**.** The agent extracted LaTeX formulas from a PDF but included markdown headers alongside the formulas in the output file, producing 6 entries instead of the required 5. The specification required one formula per line wrapped in `$$` delimiters; the extraneous headers violated this format constraint. This failure illustrates agents' tendency to over-include content rather than strictly adhering to output specifications.

Domain Knowledge Gap: `exceltable-in-ppt`**.** The agent correctly updated the primary exchange rate cell in the PowerPoint-embedded Excel table (6/8 tests passed) but failed to recompute inverse rates and dependent cells, producing `NaN` propagation through the spreadsheet. The underlying issue was misunderstanding how Excel formula dependencies cascade in embedded workbooks—a domain-specific detail that Skills could address.

Skills transforming outcomes: `sales-pivot-analysis`**.** Without Skills, Codex populated source data correctly but could not create Excel pivot tables (10/23 tests failed with "list index out of range" on missing pivot objects). With Skills, the Skills provided Office-specific guidance for programmatic pivot table creation that the agent could not discover independently. In the latest aggregate, this task improves from 1.9% to 40.7% (+38.9 pp), illustrating how Skills can bridge a specific capability gap.

J.3 Tasks With Largest Skills Impact

Table 13 lists the 10 tasks where Skills produced the largest improvement in pass rate. These tasks share a common pattern: they require domain-specific procedural knowledge (e.g., cache behavior, intrusion detection, financial reporting schemas, scientific data processing pipelines, optimization, and artifact conversion) that is well-suited to being encoded in Skill documents. The average improvement for these top-10 tasks is +67.0 percentage points.

Table 13: Tasks with largest Skills impact (With Skills pass rate − No Skills pass rate), using the latest fixed three-trial denominator.

Task	No Skills	With Skills	Δ
llm-prefix-cache-replay	1.9%	94.4%	+92.6 pp
dapt-intrusion-detection	0.0%	81.5%	+81.5 pp
sec-financial-report	0.0%	68.5%	+68.5 pp
flood-risk-analysis	1.9%	68.5%	+66.7 pp
protein-expression-analysis	11.1%	77.8%	+66.7 pp
earthquake-plate-calculation	3.7%	68.5%	+64.8 pp
software-dependency-audit	1.9%	61.1%	+59.3 pp
threejs-structure-parser	0.0%	59.3%	+59.3 pp
lake-warming-attribution	5.6%	61.1%	+55.6 pp
manufacturing-fjsp-optimization	0.0%	55.6%	+55.6 pp

K Skill Invocation Rate

Table 14 reports the exact counts behind Figure 5. Counts include only task-specific Skills shipped under each task's `environment/skills`; harness-bundled Skills are treated as part of the harness.

Table 14: Skill Invocation Rate by model–harness configuration on curated-Skills trials. Counts require a task-bundled Skill; harness-bundled Skills are excluded.

Harness	Model	Skill Invocation Rate	Invoked	Resolution Rate	Resolution | Invoked
Codex	GPT-5.5	99.2%	259/261	66.5%	66.2%
OpenHands	GPT-5.5	92.0%	240/261	67.3%	68.4%
Gemini CLI	Gemini 3.1 Pro	90.4%	236/261	60.8%	63.0%
OpenHands	Claude Sonnet 4.6	88.9%	232/261	47.2%	49.2%
OpenHands	Grok 4.3	84.7%	221/261	41.7%	46.1%
OpenHands	Claude Opus 4.8	83.9%	219/261	54.1%	54.5%
OpenHands	GLM 5.1	76.2%	199/261	58.4%	61.7%
OpenHands	Claude Opus 4.7	74.3%	194/261	53.1%	54.7%
OpenHands	MiniMax M2.7	70.5%	184/261	34.9%	40.9%
OpenHands	DeepSeek V4 Pro	69.3%	181/261	50.1%	56.8%
OpenHands	MiniMax M3	68.6%	177/258	53.0%	54.6%
Claude Code	Opus 4.7	68.2%	178/261	61.2%	63.9%
OpenHands	DeepSeek V4 Flash	64.4%	168/261	44.7%	49.8%
OpenHands	GPT-5.4 Mini	62.5%	163/261	41.4%	45.6%
OpenHands	Kimi K2.6	57.9%	151/261	54.0%	57.3%
OpenHands	Gemini 3.5 Flash	55.9%	146/261	48.2%	51.0%
OpenHands	Gemini 3.1 Pro	54.0%	141/261	52.8%	68.7%
OpenHands	Gemini 3.1 Flash Lite	46.4%	121/261	20.1%	29.8%

L Token, Cost, and Time Efficiency

This section reports token usage, per-trial cost, and agent wall-clock time for the latest 18-configuration aggregate, computed directly from the selected public `result.json` files (HF `main` + PR#11 fixed at `a26b2100`, across `v0.1` and `v1.1` submissions) over the same healthy-first three-trial selection used in the main result tables. Each trial records its own token counts and agent-execution timing; where the provider is priced through our LiteLLM runtime it also records a

per-trial cost in USD. Token counts are available for all 18 configurations and LiteLLM cost for 10 of them (several open-weight and CLI providers are not yet priced in our runtime). We average over selected trials with recorded usage or cost and suppress any cell backed by fewer than 20 usable trials.

L.1 Token Usage and Cost by Configuration

Table 15 reports mean total tokens per trial (prompt + completion, as recorded by the harness) and mean LiteLLM cost per trial, separately for the no-Skills and curated-Skills conditions.

Table 15: Mean per-trial token usage and cost in the latest 18-configuration aggregate, by Skills condition. **Tok** = mean total tokens per trial (thousands); **$** = mean LiteLLM-computed cost per trial. Cells with fewer than 20 usable trials are shown as "–"; "–" in a $ column also marks a provider not priced through our LiteLLM runtime. Claude Code counts are session-JSONL-derived and approximate.

		No Skills		Curated Skills	
Harness	**Model**	**Tok (K)**	**$**	**Tok (K)**	**$**
OpenHands	GPT-5.5	1,197	1.73	1,267	1.67
Gemini CLI	Gemini 3.1 Pro	1,165	–	1,932	–
OpenHands	GLM 5.1	2,371	–	2,117	–
Codex	GPT-5.5	3,222	3.08	3,227	2.99
Claude Code	Opus 4.7†	4,332	5.21	6,425	6.74
OpenHands	Claude Opus 4.7	1,771	10.99	1,094	6.37
OpenHands	Claude Opus 4.8	2,914	22.70	2,432	14.02
OpenHands	Gemini 3.1 Pro	1,818	0.84	4,018	1.88
OpenHands	MiniMax M3	4,718	–	4,404	–
OpenHands	Gemini 3.5 Flash	2,827	1.24	2,675	1.18
OpenHands	Kimi K2.6	2,157	–	2,103	–
OpenHands	DeepSeek V4 Pro	1,832	–	1,687	–
OpenHands	Claude Sonnet 4.6	1,834	12.34	2,874	10.15
OpenHands	Grok 4.3	489	–	579	–
OpenHands	DeepSeek V4 Flash	1,923	–	2,449	–
OpenHands	MiniMax M2.7	1,517	–	2,969	–
OpenHands	GPT-5.4 Mini	1,282	0.25	1,393	0.30
OpenHands	Gemini 3.1 Flash Lite	3,186	0.15	4,828	0.22

†Claude Code token counts are derived from session JSONL logs and are approximate (effective input includes cache-creation and cache-read tokens).

L.2 Outcome and Runtime Diagnostics

Table 16: Trial-level outcome distribution in the latest 18-configuration aggregate ($N = 9{,}396$ selected public trials), overall and by Skills condition. **Attempted** = a verifier-scored trial with reward 0; **Runtime err.** = no scored result (agent/harness/verifier error), counted as reward 0 in the main aggregate.

Condition	n	**Solved**	**Partial**	**Attempted**	**Runtime err.**
No Skills	4,698	31.3%	5.4%	62.7%	0.6%
Curated Skills	4,698	47.7%	4.7%	46.5%	1.0%
Overall	**9,396**	**39.5%**	**5.1%**	**54.6%**	**0.8%**

Table 17: Runtime errors in the latest 18-configuration aggregate (trials with no scored result; 78 of 9,396 = 0.8%). Classified from the recorded `error/error_category` fields.

Error class	Count	Source
Command timeout	72	Shell command exceeded its per-command limit
Agent-process crash	0	ACP internal/transport error
Agent timeout (wall-clock/idle)	5	Agent exceeded its wall-clock or idle budget
Subprocess crash	0	Local subprocess/transport closed unexpectedly
Verifier error	1	Verifier crash or timeout
Total	**78**	0.8% of selected public trials

L.3 Cost-Performance Tradeoff

Per-trial cost spans roughly two orders of magnitude across the lineup. The cheapest priced configurations are OpenHands + Gemini 3.1 Flash Lite ($0.15–0.22) and OpenHands + GPT-5.4 Mini ($0.25–0.30); the Gemini and GPT-5.5 configurations occupy the mid-range ($0.8–3.1 per trial); and the Claude Opus configurations are by far the most expensive—OpenHands + Claude Opus 4.8 at $14.02–22.70 per trial, OpenHands + Claude Opus 4.7 at $6.37–10.99, and OpenHands + Claude Sonnet 4.6 at roughly $10–12.3. Cost does not track capability monotonically: OpenHands + GPT-5.5 attains the highest with-Skills pass rate (67.3%) at $1.67 per trial—about eight times cheaper than OpenHands + Claude Opus 4.8 (54.1% at $14.02).

Skills do not move token usage in a single direction. Several configurations consume *more* tokens with Skills (OpenHands + Gemini 3.1 Pro 1.82M $\rightarrow$ 4.02M; OpenHands + Claude Sonnet 4.6 1.83M $\rightarrow$ 2.87M; OpenHands + Gemini 3.1 Flash Lite 3.19M $\rightarrow$ 4.83M), consistent with the agent reading and acting on the additional Skill context, while others consume *fewer* (OpenHands + Claude Opus 4.7 1.77M $\rightarrow$ 1.09M; OpenHands + Claude Opus 4.8 2.91M $\rightarrow$ 2.43M; OpenHands + Gemini 3.5 Flash 2.83M $\rightarrow$ 2.67M), consistent with Skills letting the model reach a solution with less exploratory work. Where providers expose prompt-caching fields, cache reads can account for a material share of effective input tokens, so realized cost under cached pricing can differ substantially from standard-rate estimates.

L.4 Time–Performance Tradeoff

Figure 15 plots resolution rate against mean agent wall-clock per task—the agent-execution span of each selected trial (agent runtime only), averaged within each task and then across the 87 tasks—for both Skills conditions, over the same selection as Table 10. Curated Skills lift the fleet mean by +16.6 pp at broadly unchanged agent time; the slower half of the fleet runs faster with Skills (e.g., OpenHands + Claude Opus 4.7: 10.0 $\rightarrow$ 6.5 minutes per task). Speed and capability are not in tension: the 4–9-minute band contains both the strongest and the weakest configuration, while the slowest, OpenHands + Grok 4.3, remains below the with-Skills fleet mean.

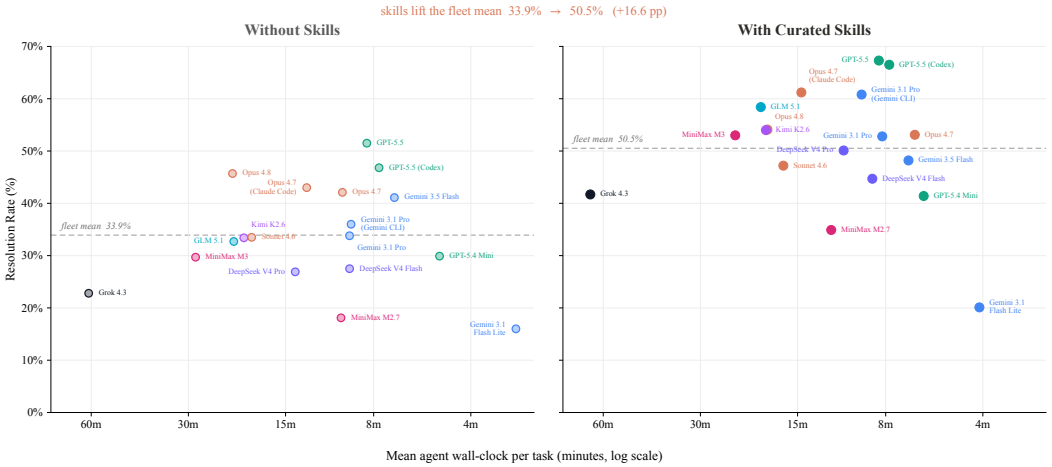

Figure 15: Resolution rate vs. mean agent wall-clock per task (minutes, log scale), without (left) and with (right) curated Skills; dashed lines mark fleet means.

M SkillsBench Construction Details

This appendix collects the construction and quality-control details summarized in §3. It is kept separate from the other appendices so that the benchmark definition, submission process, and leakage safeguards can be inspected as a single unit; the contributor-facing format and PR-level review mechanics (CI commands, checklists, report template, lifecycle) are documented in Appendix B.

M.1 Skill Definition and Boundary

A **Skill** is an artifact satisfying four operational criteria:

- **Procedural content**: Contains how-to guidance, workflows, standard operating procedures, or domain conventions rather than factual retrieval.
- **Task-class applicability**: Applies to a class of problems rather than a single benchmark instance.
- **Structured components**: Includes a `SKILL.md` file plus optional resources such as scripts, templates, references, or worked examples.
- **Portability**: Is represented as ordinary file-system content, making it easy to edit, version, share, and use across Skills-compatible agent harnesses.

This definition explicitly excludes system prompts, which lack file-structured resources; few-shot examples [Brown et al., 2020], which are primarily declarative rather than procedural; RAG retrievals [Lewis et al., 2020], which provide factual context rather than procedural workflows; and tool documentation [Schick et al., 2023, Qin et al., 2024], which describes tool capabilities rather than how to solve a task class. The boundary is not absolute–for example, a StackOverflow answer may mix factual and procedural content–but these criteria provide an operational definition for benchmark construction.

In SkillsBench, each Skill is a modular package under `environment/skills/`. The required `SKILL.md` specifies how to approach a task class, while optional resources may include executable scripts, code templates, reference documentation, or examples that the agent can invoke or consult.

M.2 Task Package

Each task is a self-contained module with four required components:

- **Instruction.** A human-readable task description specifying the objective, input format, constraints, and expected output. Instructions are written to be solvable by a knowledgeable human without access to the paired Skills, though Skills may substantially reduce time-to-solution.

- **Environment.** A Docker container with task-specific data files and a `skills/` subdirectory containing modular Skills packages. Containerization provides isolated dependencies and clean file-system state.
- **Oracle.** A reference solution demonstrating that the task is resolvable. The oracle must pass the verifier with 100% success before a task can be accepted. (The on-disk directory is `oracle/`.)
- **Verifier.** Deterministic test scripts with programmatic assertions, including numeric tolerances where appropriate. This yields reproducible pass/fail judgments without LLM-as-a-judge variance. (The on-disk directory is `verifier/`.)

M.3 Community Submission Model

To maximize domain and workflow diversity, we used a community-driven, open-source contribution model. In total, 142 contributors from academia and industry submitted 400 candidate tasks. We counted submissions that included the full task specification–instruction, environment, oracle, and verifier–along with contributor-assessed difficulty ratings. From this pool, maintainers curated the final evaluated set used in the main experiments.

M.4 Contributing Principles

Contributors were required to satisfy explicit quality requirements designed to prevent shortcuts and ensure that each task measures Skills efficacy rather than prompt-following.

Human-authored instructions. Task instructions must be written by humans, not generated by language models. We enforce this because LLM-generated queries can inherit the distributional biases of the same systems under evaluation and often lack the specificity needed for realistic agentic tasks.

Skill generality. Skills must provide procedural guidance for a *class* of tasks, not solutions to specific instances. Task instructions must not reference which Skills to use, requiring agents to discover and apply relevant Skills autonomously.

Deterministic verification. All success criteria must be testable through programmatic assertions. We target the minimal number of tests needed for verification, avoiding both insufficient coverage and redundant test bloat that can artificially depress pass rates. Tests must include informative error messages and use parametrization rather than duplication when possible.

M.5 Automated Validation

Each submission undergoes automated validation before human review:

- **Structural validation**: Required files must be present (`task.md` with `oracle/solve.sh` and `verifier/test_outputs.py`; Appendix B), and directory layout and frontmatter syntax must be valid.
- **Oracle execution**: The reference solution must achieve a 100% verifier pass rate. Tasks with failing oracles are rejected.
- **Instruction quality**: Instructions must be human-written, verified through both human review and GPTZero screening. We additionally score instructions on explicit output paths, structured requirements, success criteria, listed constraints, and context-first ordering.
- **Leakage audit**: CI checks for potential Skill-solution leakage, including task-specific constants, filenames, paths, expected outputs, and hard-coded command sequences.

M.6 Human Review

After automated checks pass, maintainers manually review each task using five criteria:

1. **Data validity**: Input data should reflect real-world complexity; synthetic or toy data is rejected unless explicitly justified.
2. **Task realism**: Scenarios should reflect realistic professional workflows without artificial difficulty.
3. **Oracle quality**: Reference solutions should match how domain experts would solve the task.

4. **Skill quality**: Skills must be error-free, internally consistent, and useful for similar tasks beyond this benchmark.
5. **Anti-cheating**: Tasks must prevent shortcut solutions such as editing input data, extracting answers from test files, or exploiting verifier implementation details.

Reviewers run benchmark experiments with and without Skills across multiple agents to confirm that each task provides meaningful signal about Skill efficacy. Figure 2 provides an end-to-end view of the benchmark construction, quality filtering, and evaluation pipeline.

M.7 Leakage Prevention

To prevent Skills from encoding task-specific solutions, we enforce explicit authoring guidelines and conduct leakage audits. A Claude Code Agent SDK-based validation agent runs in CI to detect potential Skill-solution leakage; failed tasks are rejected. Skills must not contain task-specific filenames, paths, identifiers, constants, magic numbers, values from task specifications, exact command sequences that solve benchmark tasks, references to specific test cases, or expected outputs.

Skills must apply to a class of tasks rather than a single instance, provide procedural guidance about how to approach the task class rather than declarative answers about what to output, and be authored independently of benchmark specifications. These rules are central to the paired design: if a Skill contains the answer path for one benchmark instance, the with-Skills condition would no longer measure reusable procedural augmentation.

M.8 Benchmark Composition

Table 18: Task difficulty stratification based on estimated human completion time in the accepted task pool.

Difficulty	Tasks	Human Time
Core	6 (6.9%)	$<$60 min
Extended	53 (60.9%)	1–4 hours
Extreme	28 (32.2%)	$>$4 hours

The current SKILLSBENCH task inventory comprises 87 tasks across 8 domains. Task difficulty is measured by estimated completion time for median specialists in the task domain, without assistance from AI tools. Original contributors provided time estimates, and maintainers with matching domain expertise reviewed those estimates.

M.9 Domain Taxonomy and Per-Task Mapping

We classify each released task into one of eight domains. The taxonomy uses broad top-level labels for professional context while preserving finer distinctions in task metadata. Cybersecurity and Media & Content Production are currently below $N = 8$, so per-domain inferential claims should treat those slices as descriptive until additional tasks are added.

Table 19 lists every task in the current 87-task inventory, its primary capability axis (a separate cut from the goodtask-v2 framework: Reasoning, Agentic Coding, Multimodal, Tool Use, or Search & Research), and its difficulty marker. The mapping is generated automatically from `scripts/taxonomy.csv` by `scripts/distribution.py` so it stays in sync with the figures referenced in §3.

Table 19: Per-domain task listing for the released 87-task SKILLSBENCH. Difficulty marker: **C**=Core (<60 min), **X**=Extended (1–4 h), **E**=Extreme (>4 h). Capability is the primary skill the task exercises.

Task	Capability	Diff.
Software Engineering ($n = 16$). *Code implementation, debugging, build repair, migration, testing, repo analytics.*		
`azure-bgp-oscillation-route-leak`	Reasoning	X
`data-to-d3`	Agentic Coding	X
`debug-trl-grpo`	Agentic Coding	E
`dialogue-parser`	Agentic Coding	C
`fix-build-agentops`	Agentic Coding	C
`fix-build-google-auto`	Agentic Coding	C
`fix-visual-stability`	Agentic Coding	E
`flink-query`	Agentic Coding	E
`jax-computing-basics`	Agentic Coding	X
`llm-prefix-cache-replay`	Agentic Coding	X
`parallel-tfidf-search`	Agentic Coding	X
`python-scala-translation`	Agentic Coding	X
`react-performance-debugging`	Agentic Coding	E
`simpo-code-reproduction`	Agentic Coding	E
`spring-boot-jakarta-migration`	Agentic Coding	E
`tictoc-unnecessary-abort-detection`	Reasoning	E
Industrial & Physical Systems ($n = 14$). *Power systems, manufacturing, robotics/control, construction, and physical simulation.*		
`3d-scan-calc`	Reasoning	E
`ada-bathroom-plan-repair`	Reasoning	E
`adaptive-cruise-control`	Reasoning	X
`drone-planning-control`	Reasoning	X
`dynamic-object-aware-egomotion`	Reasoning	X
`energy-ac-optimal-power-flow`	Reasoning	X
`energy-market-pricing`	Reasoning	E
`energy-unit-commitment`	Reasoning	E
`grid-dispatch-operator`	Reasoning	X
`hvac-control`	Reasoning	X
`manufacturing-codebook-normalization`	Reasoning	X
`manufacturing-equipment-maintenance`	Reasoning	X
`manufacturing-fjsp-optimization`	Reasoning	X
`r2r-mpc-control`	Reasoning	X
Natural Science ($n = 14$). *Astronomy, seismology, hydrology, materials, physics, and biomedical analysis.*		
`crystallographic-wyckoff-position-analysis`	Reasoning	X
`earthquake-phase-association`	Reasoning	E
`earthquake-plate-calculation`	Reasoning	X
`exoplanet-detection-period`	Reasoning	X
`flood-risk-analysis`	Reasoning	X
`glm-lake-mendota`	Reasoning	E

(continued on next page)

(continued from previous page)

Task	Capability	Diff.
gravitational-wave-detection	Reasoning	X
lab-unit-harmonization	Reasoning	X
lake-warming-attribution	Reasoning	X
mars-clouds-clustering	Reasoning	E
protein-expression-analysis	Reasoning	X
quantum-numerical-simulation	Reasoning	X
radar-vital-signs	Reasoning	X
seismic-phase-picking	Reasoning	E
Office & White Collar ($n = 14$). *Office documents, spreadsheets, presentations, forms, PDFs, OCR, and enterprise search.*		
citation-check	Search & Research	X
court-form-filling	Tool Use	C
edit-pdf	Tool Use	X
enterprise-information-search	Search & Research	E
exceltable-in-ppt	Tool Use	X
jpg-ocr-stat	Multimodal	E
latex-formula-extraction	Multimodal	X
offer-letter-generator	Tool Use	C
organize-messy-files	Tool Use	X
paper-anonymizer	Multimodal	X
pdf-excel-diff	Multimodal	X
powerlifting-coef-calc	Tool Use	C
pptx-reference-formatting	Tool Use	X
sales-pivot-analysis	Tool Use	X
Finance & Economics ($n = 9$). *Financial modeling, accounting, macroeconomic analysis, risk, and fraud workflows.*		
econ-detrending-correlation	Tool Use	X
financial-modeling-qa	Multimodal	E
invoice-fraud-detection	Multimodal	E
reserves-at-risk-calc	Tool Use	X
sec-financial-report	Search & Research	E
shock-analysis-demand	Tool Use	X
shock-analysis-supply	Tool Use	E
weighted-gdp-calc	Tool Use	X
xlsx-recover-data	Tool Use	X
Mathematics & OR ($n = 8$). *Formal proofs, optimization, planning, routing, scheduling, and combinatorial reasoning.*		
bike-rebalance	Reasoning	X
civ6-adjacency-optimizer	Reasoning	E
exam-block-sequencing	Reasoning	E
lean4-proof	Reasoning	X
paratransit-routing	Reasoning	E
pddl-airport-planning	Reasoning	X

(continued on next page)

(continued from previous page)

Task	Capability	Diff.
`pddl-tpp-planning`	Reasoning	X
`travel-planning`	Search & Research	X
Cybersecurity ($n = 7$). *CVE remediation, IDS, fuzzing, dependency audit, network security.*		
`dapt-intrusion-detection`	Multimodal	E
`fix-druid-loophole-cve`	Agentic Coding	E
`fix-erlang-ssh-cve`	Agentic Coding	E
`setup-fuzzing-py`	Agentic Coding	X
`software-dependency-audit`	Search & Research	X
`suricata-custom-exfil`	Multimodal	X
`syzkaller-ppdev-syzlang`	Agentic Coding	X
Media & Content Production ($n = 5$). *Video, audio, 3D content, dubbing, media transformation, and content production.*		
`mario-coin-counting`	Multimodal	X
`multilingual-video-dubbing`	Multimodal	X
`threejs-structure-parser`	Multimodal	X
`threejs-to-obj`	Multimodal	X
`video-silence-remover`	Multimodal	E

N Experimental Protocol Details

This appendix expands the concise experimental setup in §4. It is separated from the benchmark-construction appendix to keep the evaluation protocol, agent configurations, and scoring choices easy to audit.

N.1 Model–Harness Configurations

We evaluate four terminal-agent harnesses in the latest aggregate: OpenHands, Claude Code [Anthropic, 2025b], Codex CLI [OpenAI, 2025], and Gemini CLI [Google, 2025]. Each model is evaluated with the harness shown in Appendix Table 5. All models use temperature 0. The full model identifiers, harness versions, and selected result counts are listed in Appendix Table 5.

These commercial harnesses tightly couple model behavior, context construction, tool interfaces, and execution control. We therefore report results at the model–harness level rather than treating models as isolated components. Claude models have been trained with awareness of the Agent Skills specification [Anthropic, 2025a], which may affect how they interpret Skill-formatted instructions.

N.2 Skills Conditions

Each task in the latest aggregate is evaluated under two matched conditions:

- **No Skills**: The agent receives the task instruction (the `task.md` body) and task data, but no `environment/skills/` content is present.
- **Curated Skills**: The complete `environment/skills/` directory is available, including `SKILL.md` files, examples, scripts, and references.

A third condition, self-generated Skills, is evaluated on the three dedicated-harness configurations and reported separately (Appendix D.6); it is not part of the two-condition aggregate in Table 2.

N.3 Execution Protocol

For the curated-Skills condition, task Skills are exposed before the task instruction using each harness's native loading mechanism. The task environment, input data, resource limits, and verifier are otherwise identical across conditions. The agent interacts with the containerized environment until it submits a final answer or artifact. The verifier then runs deterministic assertions and writes a pass/fail result.

Selected rows first require a public result file, complete trajectory metadata, and a healthy verifier-scored pass/fail outcome; timeout rows are used only as failure backfill when healthy replacements are unavailable. Stale, rate-limited, or otherwise unscored runs are treated as incomplete coverage and rerun rather than as benchmark outcomes. Container resources, retry policy, and orchestration settings are summarized in Appendix D.

N.4 Scoring

The primary score is task-macro pass rate, following Terminal-Bench [Merrill et al., 2026]. For each task t and condition c, we average the three selected public trials:

$$s_{t,c} = \frac{1}{3} \sum_{i=1}^{3} r_{t,c,i}.$$

where $r_{t,c,i} \in [0, 1]$ is the deterministic verifier reward for trial i. We then average task scores over the fixed set of 87 tasks:

$$\mathrm{PassRate}(c) = \frac{1}{87} \sum_{t=1}^{87} s_{t,c}.$$

The main no-Skills and curated-Skills conditions use this fixed three-trial denominator per task–model pair. We report absolute improvement, $\Delta = \mathrm{PassRate}(\text{curated}) - \mathrm{PassRate}(\text{no Skills})$, and normalized gain:

$$g = \frac{\mathrm{PassRate}(\text{curated}) - \mathrm{PassRate}(\text{no Skills})}{1 - \mathrm{PassRate}(\text{no Skills})}.$$

Normalized gain measures proportional progress toward perfect performance, but it can overstate small absolute gains near the ceiling. We therefore interpret g jointly with absolute pass-rate deltas. The confidence intervals shown in Figure 1 use the binomial calculation described in Appendix G.