From Raw Experience to Skill Consumption: A Systematic Study of Model-Generated Agent Skills

Zisu Huang[1,2,*,†] **Jingwen Xu**[1,*] **Yifan Yang**[2,‡] **Ziyang Gong**[3] **Qihao Yang**[3]
Muzhao Tian[1] **Xiaohua Wang**[1] **Changze Lv**[1] **Xuemei Gao**[2] **Qi Dai**[2] **Bei Liu**[2]
Kai Qiu[2] **Xue Yang**[3] **Dongdong Chen**[2] **Xiaoqing Zheng**[1,‡] **Chong Luo**[2]
[1]Fudan University [2]Microsoft Research [3]Shanghai Jiao Tong University
yifanyang@microsoft.com, zhengxq@fudan.edu.cn

Abstract

Language agents increasingly improve by reusing *skills*—structured procedural artifacts distilled from past experience. In particular, *domain-level* and *model-generated* skills are especially promising. They offer fast adaptation within a domain by encoding domain-specific recurring procedures, and they scale beyond labor-intensive hand-crafting. However, while extraction methods continue to proliferate, understanding remains limited, with no comprehensive study spanning the full skill lifecycle—**experience generation**, **skill extraction**, and **skill consumption**—to ask whether such skills actually work, when they work, and what makes them succeed or fail. To close this gap, we build a utility-grounded evaluation framework that provides systematic experimental results across extractors and target agents, covering five diverse agentic task domains. We find that model-generated skills are beneficial on average but exhibit non-trivial negative transfer, and that neither extractors nor targets behave uniformly. A model can be a strong extractor yet a weak consumer, or vice versa, with skill utility independent of model scale or baseline task strength. To explain these patterns, we then dissect each lifecycle stage in depth, analyzing how experience composition shapes skill quality, what properties characterize useful skills, and how the same skill transfers across different consumers. Finally, we translate these findings into a concrete *meta-skill* that guides skill extraction toward the features tied to actual utility, which consistently improves skill quality across domains and substantially reduces negative transfer. Code is available at https://aka.ms/SkillLens.

1 Introduction

Language agents increasingly improve by reusing knowledge distilled from past trajectories: *skills*—short, structured procedural artifacts—can be loaded at inference time without retraining and have become a defining mechanism for accumulating experience in modern agent stacks [1, 2]. In particular, *domain-level skills* package a domain's recurring procedures into a single reusable artifact or a coordinated set of them, enabling fast adaptation to new tasks within the domain rather than per-task optimization. As the practical value of hand-crafted skills has been progressively demonstrated in real-world deployments, skills have become a standard component in several commercial agent platforms [3]. However, hand-crafting skills is labor-intensive and cannot keep pace with the rapidly expanding scope of agent capabilities and deployment.

Therefore, a growing literature turns to *model-generated skills*, producing them automatically at scale [4–8], with featured works either directly distilling them from execution logs as in

[*]Equal contribution. [†]Work done during an internship at MSRA. [‡]Corresponding authors.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

Trace2Skill [9], or iteratively refining multi-file skill packages with a co-evolving verifier as in CoEvoSkills [10]—offering scalability and automated iteration for agent skills. At their core, all these methods follow the same skill lifecycle: generating execution trajectories through agent–environment interaction (**experience generation**), extracting reusable knowledge or patterns from them (**skill extraction**), and consuming the resulting skills at inference time (**skill consumption**). Despite this methodological momentum, evaluation and understanding lag behind. Recent benchmarks each illuminate one slice of the picture but leave the lifecycle as a whole opaque. Most existing efforts study only the skill consumption stage, measuring the marginal performance gain from skill equipment: SkillsBench [11] uses task-seeded, human-authored skills, while SWE-Skills-Bench [12] and Skills-in-the-Wild [13] draw skills from existing public skill repositories instead—all leaving the skill extraction stage outside the loop. A notable step toward studying the skill extraction stage is SkillCraft [14], which extracts skills as executable compositions of atomic tools and studies their reuse across tasks. However, it has notable limitations: skills are restricted to executable function compositions, and the benchmark's tasks are designed and scaled to admit such compositions, making it unclear whether the paradigm generalizes to broader domains whose tasks are not designed around function-style reuse. Taken together, these efforts leave a clear gap: no comprehensive study examines all three stages of the skill lifecycle and systematically asks whether domain-level, model-generated skills actually work, when they work, and what makes them work or fail.

To close this gap, we conduct a comprehensive, utility-grounded study of model-generated, domain-level skills across all three stages of the skill lifecycle. Specifically, we follow a three-step pipeline: a target agent first executes an experience-generation split to produce an experience pool; an extractor then distills this pool into a single domain-level skill through an extraction framework with minimal design, reflecting the extractor's own ability rather than scaffolding tricks; the resulting skill is finally applied back to the same target and evaluated on the held-out test split to obtain the performance change relative to a no-skill baseline, which we use as a proxy for skill utility. We instantiate this pipeline across five domains, spanning embodied planning, productivity software, software engineering, web search, and tool calling, and systematically vary the extractor and target. Based on these experiments, we further introduce two metrics that disentangle the two roles: the **Extraction Efficacy** (EE)—how reliably a fixed extractor produces helpful skills across targets—and the **Target Evolvability** (TE)—how much a fixed target benefits from skills extracted by different extractors from its own experience. Beyond reporting these metrics, we further provide an in-depth analysis spanning all three lifecycle stages, aiming to explain the observed utility patterns and to point toward concrete directions for improving skill extraction. The pipeline and analysis are summarized in Figure 1. Overall, our study is organized around three research questions:

- **RQ1** Do model-generated, domain-level skills reliably benefit downstream agents across targets, extractors, and domains? (Section 4)
- **RQ2** Across the three lifecycle stages of experience generation (Section 5.1), skill extraction (Section 5.2), and skill consumption (Section 5.3), what actually drives a skill's downstream utility?
- **RQ3** Can the empirical findings in our study be transformed into a concrete, drop-in improvement to skill extraction itself? (Section 6)

Answering these questions, we aim to move the entire skill lifecycle from heuristic, intuition-driven practice toward a principled, utility-grounded discipline. As skill libraries proliferate across heterogeneous models and domains, our study helps practitioners obtain skills that are genuinely stable and effective when deployed in real agent systems.

2 Related Work

Automatic Generation of Reusable Knowledge from Agent Experience. Recent work explores mechanisms for extending LLM-agent capabilities without retraining [1, 15]. Agent skills—composable packages of instructions, code, and resources loaded on demand—provide one such mechanism, motivating a growing body of work on automatically extracting such skills from execution trajectories. These methods scale with collected experience, transfer across tasks and environments, and largely organize around trajectory-to-skill extraction as the core primitive. Prior agents derive reusable skills, insights, and workflows from trajectories [16–18]. *Prompt-based distillation* methods directly summarize trajectories into structured skill artifacts: Trace2Skill [9] employs parallel sub-agents followed by hierarchical consolidation, AutoRefine [4] induces dual-form experience

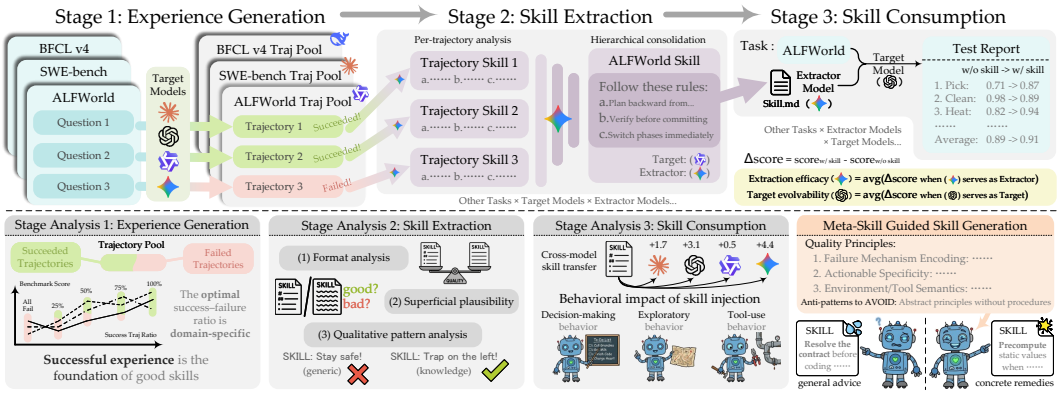


Figure 1: Overview of our study design. We evaluate the full trajectory-to-skill lifecycle across three stages: experience generation, skill extraction, and skill consumption.

patterns, PRAXIS [5] builds state-indexed procedural memory, and MemP [19] formalizes the build–retrieve–update cycle of agent procedural memory. *Optimization and RL-based methods* further refine extracted skills: ProcMem [6] applies non-parametric PPO, CoEvoSkills [10] uses co-evolutionary verification, SkillOpt [20] optimizes skill documents through trajectory-driven edits and validation-gated updates, and others combine skill banks with reinforcement learning [7, 8, 21]. A third line studies *self-evolving lifecycle agents* that iteratively refine skills through closed-loop deployment, as in EvolveR [22]. Despite their differences, all of these approaches rely on trajectory-to-skill extraction as the foundational step that turns raw agent experience into reusable knowledge. Complementarily, Resource2Skill distills human-created multimodal resources into executable skills [23]. While these works propose effective extraction methods, they each operate under their own setup, and do not provide a systematic understanding spanning the full experience–extraction–consumption lifecycle; our study addresses both gaps through systematic variation across extractors, target models, and domains together with stage-by-stage analysis.

Benchmarks for Agent Skills. Recent benchmarks probe complementary aspects of the agent-skill landscape. One group focuses on *whether skills help at all*: SkillsBench [11], SWE-Skills-Bench [12], and Skills-in-the-Wild [13] primarily test whether curated or discovered skills improve downstream performance over a no-skill baseline. Another emphasizes *retrieval and orchestration at scale*: AgentSkillOS [24] studies ecosystem-level skill management, while SkillFlow [25] develops scalable retrieval over large skill repositories. Most closely related to our setting, SkillCraft [14] studies *composition and accumulation* via an extraction-and-reuse protocol at test time; however, it restricts skills to executable functions, limiting the diversity of skill representations explored. Despite this rapid progress, the field still lacks a systematic understanding of the full trajectory-to-skill lifecycle across the raw experience generation, skill extraction, and skill consumption stages. We address this gap with a comprehensive evaluation framework that crosses skill extractors, skill consumers, and domains, accompanied by detailed analysis of each lifecycle stage.

3 Evaluation Framework

3.1 Skill Lifecycle Formulation

Let M denote a *target model* that both generates experience and consumes skills, and let E denote a (possibly different) *extractor model*. The skill generation lifecycle consists of three stages.

Stage 1: Experience generation. In domain $\mathcal{D}$, target model M executes tasks from the training split $Q_{\mathcal{D}}^{\text{train}}$, producing an experience pool $\mathcal{T}_{M,\mathcal{D}} = \{(\text{task}_i, \text{trajectory}_i, \text{outcome}_i)\}$ containing both successful and failed trajectories.

Stage 2: Skill extraction. E distills the experience pool into a skill set $\mathcal{S}_{E,M,\mathcal{D}} = E(\mathcal{T}_{M,\mathcal{D}})$ using the extraction framework described in Section 3.2. The output is structured procedural knowledge under a fixed schema and budget constraint.

Stage 3: Skill consumption. The same target M is provided with $\mathcal{S}_{E,M,\mathcal{D}}$ and evaluated on held-out tasks $Q_{\mathcal{D}}^{\text{test}}$, measuring how well the extracted skills generalize to unseen tasks in $\mathcal{D}$.

This protocol simulates a deployment-realistic, extractor-assisted single-step evolution: skills are distilled from M's own interaction logs and fed back to the same model on held-out tasks, grounding the skill source in M's actual behavior and failure modes. Holding M fixed while varying only E enables a controlled comparison of how different extraction procedures convert a model's experience into downstream gains. The consumption analyses in Section 5.3 instead hold the skill fixed and vary the consumer independently of the experience source.

3.2 Extraction Framework

All experiments in our study use a unified extraction framework with intentionally minimal structure: no domain-specific heuristics, filtering rules, or optimization tricks, leaving all abstraction decisions to the extractor model itself. The only imposed organization is a two-stage decomposition that borrows the high-level structure of Trace2Skill [9] but strips away its sub-agent fleet, conflict resolution, and skill-deepening mechanisms, retaining only the bare per-trajectory extraction and hierarchical merging steps. This minimal design ensures that performance differences are attributable to extractor capability rather than pipeline engineering.

Per-trajectory analysis. The extractor E processes each trajectory τ_i in the experience pool independently, producing a *pattern set* u_i containing multiple success and failure patterns (up to K per trajectory):

$$E : \tau_i \longmapsto u_i = \{p_1, \ldots, p_k\}, \qquad U = \{u_1, \ldots, u_n\} \tag{1}$$

Each pattern captures a reusable behavioral insight: *success patterns* encode strategies that led to task completion, while *failure patterns* encode error modes and pitfalls. Since trajectories are processed independently, this phase is fully parallelizable.

Hierarchical consolidation. The extractor E then consolidates the pattern sets in a tree-structured reduction with configurable group size G: at each level, E merges G pattern sets by deduplicating, generalizing, and reconciling overlapping patterns until a single consolidated pattern set remains:

$$\begin{aligned} U^{(0)} &= U, \\ U^{(\ell+1)} &= \left\{ \textsc{Merge}_E\left(u^{(\ell)}_{G(j-1)+1}, \ldots, u^{(\ell)}_{Gj}\right) \right\}_j, \\ &\quad \text{until } |U^{(L)}| = 1 \end{aligned} \tag{2}$$

Finally, E converts the consolidated pattern set into the skill set $\mathcal{S}_{E,M,\mathcal{D}}$ via structured tool-calling operations that support creation, update, and deletion of skills with schema validation.

Skill representation. Each skill follows a fixed schema based on the Agent Skills open standard[1], with fields for `name`, `description`, `body` (Markdown procedural instructions), and optional `references` and `scripts`.

3.3 Evaluation Metric

We evaluate the effectiveness of extracted skills by downstream performance gain rather than text quality. For each extractor–target–domain triple $(E, M, \mathcal{D})$, we measure the performance delta caused by injecting the extracted skill:

$$\Delta(E, M, \mathcal{D}) \;=\; \text{Perf}(M \mid \mathcal{S}_{E,M,\mathcal{D}},\; Q_{\mathcal{D}}^{\text{test}}) \;-\; \text{Perf}(M \mid Q_{\mathcal{D}}^{\text{test}}) \tag{3}$$

where Perf is the domain-specific task metric. Baseline and skill-augmented evaluations use the same held-out split $Q_{\mathcal{D}}^{\text{test}}$. $\Delta > 0$ indicates improvement and $\Delta < 0$ indicates negative transfer.

For each domain, varying E and M yields the set $\{\Delta(E, M, \mathcal{D}) : E \in \mathcal{E}, M \in \mathcal{M}\}$, where $\mathcal{E}$ is the set of extractors and $\mathcal{M}$ is the set of target models. We summarize these extractor–target performance gains from two complementary perspectives for deeper insights:

Extraction efficacy. This metric captures the extractor-side effect. For a fixed extractor, it asks how reliably that extractor converts different target-specific experience pools into skills that improve

[1] https://github.com/agentskills/agentskills

downstream performance:

$$\mathrm{EE}(E, \mathcal{D}) = \frac{1}{|\mathcal{M}|} \sum_{M \in \mathcal{M}} \Delta(E, M, \mathcal{D}). \tag{4}$$

Target evolvability. This metric captures the target-side effect. For a fixed target, it asks how much the target improves when different extractors distill skills from the target's own experience and feed them back to the same target:

$$\mathrm{TE}(M, \mathcal{D}) = \frac{1}{|\mathcal{E}|} \sum_{E \in \mathcal{E}} \Delta(E, M, \mathcal{D}). \tag{5}$$

We report both EE and TE per domain, since task metrics and difficulty are domain-specific. We also retain each extractor–target Δ to analyze interactions beyond these averages.

4 Main Experiments

In this section, we conduct a large-scale evaluation of model-generated agent skills across five domains, six target models, and five extractor models. The goal is to characterize when extracted skills improve downstream performance, when they fail or degrade it, and how these outcomes vary across the extractor–target–domain space. We report the main empirical patterns here and leave deeper analysis to Section 5.

4.1 Experimental Setup

Domains. To obtain a comprehensive view of model-generated skills, our evaluation spans five qualitatively different domains: embodied interaction, productivity software, software engineering, web search, and tool calling. This breadth lets us test whether extracted skills remain useful across different forms of agent behavior:

- **ALFWorld** [26]: embodied household tasks requiring physical commonsense, exploration, and multi-step planning.
- **SpreadsheetBench** [27]: spreadsheet manipulation tasks involving table inspection, formula reasoning, filtering, and value editing.
- **SWE-bench-Verified** [28]: real-world software engineering tasks requiring codebase understanding, fault localization, and patch generation.
- **SEAL-0** [29]: web-search question answering tasks requiring retrieval, evidence synthesis, and multi-hop reasoning.
- **BFCL-v4** [30]: tool-calling tasks requiring function selection, parameter extraction, type matching, and multi-turn tool use. We use the *multi-turn* subset, which exercises long-horizon, procedural tool-use behaviour relevant to skill reuse.

Models. We select models spanning different families and scales: **GPT** (GPT-5.4, GPT-5.4-mini) [31], **Gemini** (Gemini-3.1-Pro [32], Gemini-3.1-Flash-Lite [33]), and **Qwen** (Qwen3.5-35B, Qwen3.5-9B) [34]. All six models serve as targets. During preliminary experiments, we found that Qwen3.5-9B cannot reliably follow the structured extraction protocol (Section 3.2), so it is excluded as an extractor.

Data splits and evaluation protocol. For each domain $\mathcal{D}$, we use an experience-generation split $Q_{\mathcal{D}}^{\text{train}}$ and a separate held-out test split $Q_{\mathcal{D}}^{\text{test}}$, generally in a 1:1 ratio; benchmark-specific exceptions are in Appendix B. Each target M runs $Q_{\mathcal{D}}^{\text{train}}$ to form an experience pool $\mathcal{T}_{M,\mathcal{D}}$. Each extractor E distills this pool into a single consolidated skill $\mathcal{S}_{E,M,\mathcal{D}}$ in our main experiments, which is supplied in the target's system prompt at inference time and evaluated on $Q_{\mathcal{D}}^{\text{test}}$. We run each evaluation three times and report the average Δ (Eq. 3) and its across-run standard deviation in percentage points. Full extraction and evaluation details are in Appendix B; task counts and statistical procedures are in Appendix F.

Table 1: Skill-induced gains (pp), shown as $\text{mean}_{\pm\text{SD}}$ over three inference runs. Base gives no-skill accuracy; EE and TE report extractor- and target-side average gains. Stars mark unadjusted paired $p < 0.05$; details are in Appendix F.

Target	Base	5.4	5.4-mini	3.1-Pro	3.1-FL	3.5-35B	TE
Embodied: *ALFWorld*							
5.4	68.66	$+1.49_{\pm 1.29}$	$+6.47_{\pm 0.86}^{*}$	$+7.46_{\pm 0.75}^{*}$	$+4.98_{\pm 1.55}^{*}$	$+4.23_{\pm 1.72}^{*}$	**+4.93**
5.4-mini	52.24	$+1.00_{\pm 0.43}^{*}$	$+4.23_{\pm 0.43}^{*}$	$+2.74_{\pm 0.43}^{*}$	$+2.24_{\pm 1.29}$	$+3.98_{\pm 2.15}^{*}$	+2.84
3.1-Pro	87.56	$+0.50_{\pm 0.43}$	$+0.75_{\pm 0.75}$	$+0.00_{\pm 1.29}$	$-0.75_{\pm 0.75}$	$-1.24_{\pm 1.14}$	-0.15
3.1-FL	51.99	$-2.49_{\pm 1.88}^{*}$	$-1.24_{\pm 0.43}^{*}$	$+1.49_{\pm 0.00}^{*}$	$-2.49_{\pm 0.86}^{*}$	$-3.23_{\pm 1.72}^{*}$	-1.59
3.5-35B	57.21	$-1.99_{\pm 1.55}^{*}$	$-3.48_{\pm 1.14}^{*}$	$-0.75_{\pm 1.29}$	$+0.50_{\pm 1.55}$	$-1.00_{\pm 0.43}^{*}$	-1.34
3.5-9B	36.07	$-2.49_{\pm 0.43}^{*}$	$-2.99_{\pm 0.00}^{*}$	$-1.24_{\pm 1.55}$	$-1.99_{\pm 1.14}^{*}$	$+0.25_{\pm 0.86}$	-1.69
EE		-0.66	+0.62	**+1.62**	+0.42	+0.50	
Productivity: *SpreadsheetBench*							
5.4	37.17	$+4.33_{\pm 1.04}^{*}$	$+9.00_{\pm 1.00}^{*}$	$+14.00_{\pm 1.00}^{*}$	$+14.67_{\pm 0.29}^{*}$	$+6.33_{\pm 1.44}^{*}$	**+9.67**
5.4-mini	29.33	$+0.33_{\pm 0.58}$	$+2.50_{\pm 0.00}^{*}$	$+3.67_{\pm 0.29}^{*}$	$+4.50_{\pm 0.50}^{*}$	$+1.00_{\pm 0.50}$	+2.40
3.1-Pro	35.83	$-0.50_{\pm 0.87}$	$-2.67_{\pm 0.76}^{*}$	$+6.50_{\pm 0.50}^{*}$	$+5.33_{\pm 0.76}^{*}$	$+5.83_{\pm 0.76}^{*}$	+2.90
3.1-FL	25.00	$+2.67_{\pm 1.04}^{*}$	$+1.83_{\pm 0.76}^{*}$	$+1.50_{\pm 0.50}^{*}$	$+6.17_{\pm 0.76}^{*}$	$+7.33_{\pm 0.58}^{*}$	+3.90
3.5-35B	23.83	$+2.00_{\pm 1.00}^{*}$	$+5.50_{\pm 0.50}^{*}$	$+0.17_{\pm 0.29}$	$+3.33_{\pm 1.15}^{*}$	$-3.50_{\pm 1.00}^{*}$	+1.50
3.5-9B	13.67	$+1.17_{\pm 1.04}$	$+3.17_{\pm 0.58}^{*}$	$-1.17_{\pm 0.76}$	$+1.17_{\pm 0.29}^{*}$	$+3.00_{\pm 1.00}^{*}$	+1.47
EE		+1.67	+3.22	+4.11	**+5.86**	+3.33	
Coding: *SWE-bench-Verified*							
5.4	68.40	$+4.67_{\pm 1.97}^{*}$	$+1.33_{\pm 1.01}^{*}$	$+2.00_{\pm 1.44}^{*}$	$+4.00_{\pm 1.83}^{*}$	$+2.27_{\pm 1.29}^{*}$	+2.85
5.4-mini	59.73	$+3.20_{\pm 1.44}^{*}$	$+3.20_{\pm 2.12}^{*}$	$+1.73_{\pm 2.01}$	$+3.60_{\pm 0.80}^{*}$	$+2.80_{\pm 1.06}^{*}$	**+2.91**
3.1-Pro	66.53	$+2.00_{\pm 0.40}^{*}$	$+2.80_{\pm 2.00}^{*}$	$+2.13_{\pm 1.97}^{*}$	$+3.47_{\pm 1.80}^{*}$	$-1.60_{\pm 0.00}^{*}$	+1.76
3.1-FL	55.47	$+2.67_{\pm 0.46}^{*}$	$+3.33_{\pm 0.61}^{*}$	$+2.93_{\pm 0.83}^{*}$	$+3.47_{\pm 3.00}^{*}$	$-0.93_{\pm 2.31}$	+2.29
3.5-35B	52.93	$+3.20_{\pm 1.74}^{*}$	$+2.00_{\pm 2.12}^{*}$	$+2.53_{\pm 0.46}^{*}$	$+2.93_{\pm 0.61}^{*}$	$+2.00_{\pm 0.40}^{*}$	+2.53
3.5-9B	33.33	$-1.07_{\pm 2.27}$	$+2.40_{\pm 0.40}^{*}$	$-1.60_{\pm 2.80}^{*}$	$+1.20_{\pm 2.88}$	$+0.93_{\pm 2.54}$	+0.37
EE		+2.45	+2.51	+1.62	**+3.11**	+0.91	
Web search: *SEAL-0*							
5.4	51.19	$+6.55_{\pm 1.03}^{*}$	$+4.17_{\pm 1.03}^{*}$	$+7.74_{\pm 1.03}^{*}$	$+1.79_{\pm 1.79}$	$+1.79_{\pm 1.79}$	+4.41
5.4-mini	45.24	$-1.19_{\pm 1.03}$	$+2.98_{\pm 2.06}$	$-4.17_{\pm 1.03}^{*}$	$+4.17_{\pm 1.03}^{*}$	$-4.17_{\pm 1.03}^{*}$	-0.48
3.1-Pro	55.95	$-4.17_{\pm 1.03}^{*}$	$-1.79_{\pm 1.79}$	$+1.79_{\pm 0.00}^{*}$	$+2.38_{\pm 2.06}$	$-3.57_{\pm 0.00}^{*}$	-1.07
3.1-FL	14.88	$+9.52_{\pm 2.06}^{*}$	$+8.33_{\pm 1.03}^{*}$	$+2.98_{\pm 1.03}^{*}$	$-1.19_{\pm 2.06}$	$+7.14_{\pm 0.00}^{*}$	**+5.36**
3.5-35B	40.48	$+1.79_{\pm 1.79}$	$+6.55_{\pm 1.03}^{*}$	$-3.57_{\pm 1.79}^{*}$	$+4.76_{\pm 1.03}^{*}$	$+2.38_{\pm 1.03}^{*}$	+2.38
3.5-9B	33.93	$+10.71_{\pm 0.00}^{*}$	$+8.93_{\pm 1.79}^{*}$	$-5.95_{\pm 1.03}^{*}$	$+5.95_{\pm 1.03}^{*}$	$-2.98_{\pm 1.03}^{*}$	+3.33
EE		+3.87	**+4.86**	-0.20	+2.98	+0.10	
Tool calling: *BFCL-v4*							
5.4	51.68	$+3.08_{\pm 1.48}^{*}$	$+5.04_{\pm 1.11}^{*}$	$+0.42_{\pm 1.68}$	$+5.04_{\pm 0.42}^{*}$	$-2.24_{\pm 2.07}^{*}$	+2.27
5.4-mini	53.50	$+3.92_{\pm 1.28}^{*}$	$+6.16_{\pm 3.10}^{*}$	$+7.56_{\pm 2.91}^{*}$	$+5.18_{\pm 0.49}^{*}$	$+2.94_{\pm 0.42}^{*}$	**+5.15**
3.1-Pro	51.82	$+5.32_{\pm 0.64}^{*}$	$+5.88_{\pm 3.33}^{*}$	$-4.34_{\pm 1.21}^{*}$	$+0.14_{\pm 3.88}$	$+6.02_{\pm 1.89}^{*}$	+2.60
3.1-FL	41.18	$+4.06_{\pm 0.87}^{*}$	$+4.62_{\pm 0.73}^{*}$	$+4.20_{\pm 1.68}^{*}$	$+8.12_{\pm 1.70}^{*}$	$+3.64_{\pm 2.57}^{*}$	+4.93
3.5-35B	57.56	$-0.70_{\pm 2.67}$	$+0.84_{\pm 4.54}$	$+1.54_{\pm 3.40}^{*}$	$+2.80_{\pm 1.06}^{*}$	$+1.82_{\pm 0.87}^{*}$	+1.26
3.5-9B	46.78	$+2.94_{\pm 2.62}^{*}$	$+2.94_{\pm 1.26}^{*}$	$+1.82_{\pm 2.16}^{*}$	$-0.98_{\pm 1.89}$	$-1.12_{\pm 3.26}$	+1.12
EE		+3.10	**+4.25**	+1.87	+3.38	+1.84	

4.2 Main Results

The following results answer **RQ1**: whether model-generated, domain-level skills reliably benefit downstream agents across targets, extractors, and domains. We report per-cell performance deltas across the extractor–target matrix together with the aggregated EE and TE metrics.

Model-generated skills are generally beneficial, but not guaranteed. Table 1 presents the full Δ matrix across domains. Model-generated skills are generally effective, improving downstream performance in 112/150 entries. Yet negative transfer remains common: 37/150 entries have $\Delta < 0$, meaning that applying extracted skills degrades the target's performance. This risk is domain-dependent: SpreadsheetBench and SWE-bench-Verified have the lowest negative rates (13%), whereas ALFWorld is the most fragile domain (47%). Thus, positive average gains mask a substantial risk of negative transfer, so model-generated skills cannot be assumed to improve performance.

Better executor is not necessarily better extractor. Extractor-side performance does not simply follow model scale or baseline task strength. For example, on SpreadsheetBench, the lightweight Gemini-3.1-Flash-Lite achieves the highest EE, while GPT-5.4 ranks last despite having the strongest baseline among the targets. This reversal shows that skill extraction is a distinct capability from task execution: the extractor must convert target-specific trajectories into procedural guidance that the target can actually exploit. Consequently, choosing an extractor is not equivalent to choosing the strongest model; it is a compatibility problem between extractor, target, and domain.

Skill utility is target-dependent. Even within the same domain, the same set of extractors can produce very different gains across targets. On ALFWorld, GPT-5.4 benefits consistently from all five extractors (TE $= +4.93$), while Gemini-3.1-Flash-Lite, Qwen3.5-35B, and Qwen3.5-9B all have negative TE. Similar asymmetries appear across other domains. This suggests that skill benefit is shaped not only by extractor quality, but also by what a target's own experience makes extractable and what the target can execute from the resulting guidance.

Interactive consumption and skill libraries. We also evaluate on-demand skill discovery under Codex, crossing GPT-5.4, GPT-5.4-mini, and Gemini-3.1-Pro as extractors and targets across all five domains. The 45 configurations yield 37 positive, six negative, and two zero gains; GPT-5.4-mini has the highest extractor-averaged gain. For libraries of 1, 5, and 10 skills, each model acts as both extractor and consumer. Ten skills perform best in 11/15 model–domain combinations, while the other four favor fewer skills. The visualizations, full results, and setup details are in Appendix G.

> **Finding (RQ1).** Model-generated, domain-level skills help on average but are *not* universally beneficial: 25% of evaluated extractor–target pairings exhibit negative transfer. An extractor's task-solving capability does not predict its extraction quality, and TE varies sharply across targets even within a single domain—both the extractor and the target jointly shape skill utility.

5 Diving Deeper into the Agent Skill Lifecycle

This section addresses **RQ2**: *what actually drives a skill's downstream utility?* Following the lifecycle defined in Figure 1, we further analyze the three stages separately—**experience generation**, **skill extraction**, and **skill consumption**, and ask what factors at each stage govern downstream gains.

5.1 Experience Generation: Success or Failure, Which Teaches Better Skills?

The first stage determines what information is available for extraction. A natural and key factor is the success/failure composition of the experience pool: successful trajectories expose workable procedures, while failures may expose constraints and pitfalls. We isolate this factor by directly manipulating pool composition.

Setup. We fix the extractor (GPT-5.4-mini) and sample five experience pools from the same source trajectories, with success ratios of 100%, 75%, 50%, 25%, and 0%. Each pool is converted into a skill using the same extraction pipeline. We evaluate the resulting skills across all five domains with three targets—GPT-5.4, Gemini-3.1-Pro, and Qwen3.5-35B—and report their equally weighted average Δ in Figure 2(a).

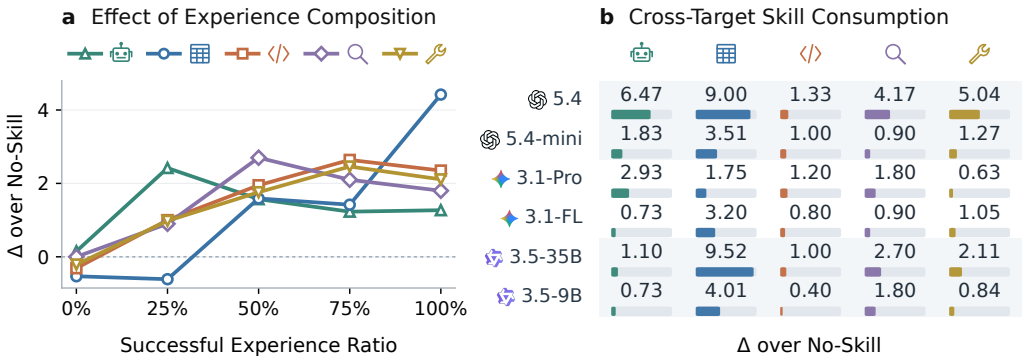


Figure 2: Two stages of the skill lifecycle across five domains (icons defined in Section 4.1). (a) Gains as the experience pool's success ratio varies, with GPT-5.4-mini as extractor. (b) Gains when the same domain-specific skill, extracted by GPT-5.4-mini from GPT-5.4 experience, is applied to six consumers. All gains are in percentage points over the no-skill baseline; the bars in (b) share a 0–10 scale.

Results. Figure 2(a) shows that **experience composition strongly affects extracted skill quality**. Beyond this, **the optimal success–failure ratio is domain-specific**. SpreadsheetBench favors all-success experience, SWE-bench-Verified and BFCL-v4 peak at 75% success, SEAL-0 at 50%, and ALFWorld at 25%. This suggests that domain-specific behavior patterns shape the informational value of successes versus failures for skill extraction: in ALFWorld, for example, failed attempts often reveal invalid actions and dead-end states, making failures surprisingly informative. Overall, Figure 2(a) also shows that all-failure pools are generally weak, highlighting **successful trajectories as the foundation of skill extraction**: they provide positive procedural signals that guide the agent's actions and narrow its exploration space, rather than merely indicating what to avoid. They are not uniformly the worst: SpreadsheetBench is slightly lower at 25% success than at 0%.

Finding (Experience). Skill quality appears to be largely influenced by the success–failure composition of the experience pool. Pure-failure pools are generally weak, while the optimal mix tends to be domain-dependent—reflecting how each domain weighs positive procedural signals against negative constraint signals.

5.2 Skill Extraction: What Makes a Good Skill?

Given that experience quality matters (Section 5.1), we now test whether skill format or textual plausibility explains utility, then examine a qualitative pattern motivating Section 6.

Skill quality is not reducible to surface form. Rewriting the same skill as an *ordered list*, *unordered list*, *checklist*, or *prose* yields no significant format effect (Friedman tests, all $p>0.34$), whereas extractor choice matters on 3/4 targets ($p<0.01$; Table 8, Appendix C). Utility depends on what a skill says, not how it looks.

Textual plausibility does not predict skill utility. If content matters, can we identify better skills from the text alone? A judge sees only two skill texts and selects the one expected to yield higher downstream performance. On held-out pairs, Figure 5 shows rubric-free accuracy of 39–53% overall, dropping to 14–40% at $\delta\geq 5$ pp; utility-grounded guidance reaches 66–75% for the latter. The skill that reads better is often the one that performs worse. Textual plausibility therefore does not reliably predict utility, motivating the utility-grounded criteria in Section 6 (protocols in Appendix E).

A qualitative hint: concrete remedies, not generic advice. A qualitative inspection of one high-δ pair in SpreadsheetBench (Table 18, Appendix H) hints at where the gap lies. The higher-Δ skill names concrete failure mechanisms with executable remedies (e.g., precomputing static values when host engines do not evaluate formula strings), whereas the lower-Δ skill offers only generic procedural advice (e.g., "resolve the contract before coding"). We treat this as motivation; Section 6 tests at scale whether guidance along such textual dimensions improves utility.

Finding (Extraction). Neither skill format nor textual plausibility reliably predicts utility; utility-grounded criteria reveal predictive signal, motivating the meta-skill analysis in Section 6.

5.3 Skill Consumption: How Does Skill Benefit Vary Across Target Models?

Sections 5.1–5.2 focus on the *supply side* of skills: what experience to extract from and what textual properties matter. Here we turn to the *demand side*: given an identical skill, how much benefit does each consumer actually derive?

Cross-model skill transfer. To isolate the consumer's contribution, GPT-5.4-mini extracts one skill from GPT-5.4 experience in each of the five domains; each skill is applied unchanged to all six targets. Figure 2(b) shows that **skill consumption ability varies across targets**: on SpreadsheetBench, the same skill yields $+1.75$ pp on Gemini-3.1-Pro but $+9.52$ pp on Qwen3.5-35B. Averaged across domains, consumer gains range from $+1.34$ pp for Gemini-3.1-Flash-Lite to $+5.20$ pp for GPT-5.4. Thus, utility is not a property of skill text alone, but also depends on the target that consumes it.

Experience-pool quality. To further examine source-pool quality, we fix GPT-5.4-mini on SpreadsheetBench and compare a *strong-pool skill* from the strongest baseline model's experience (GPT-5.4) with a *weak-pool skill* from the weakest (Qwen3.5-9B). Both skills are applied to all six targets. The strong-pool skill improves every target, whereas the weak-pool skill causes negative transfer on GPT-5.4 (-2.0 pp) and Gemini-3.1-Pro (-1.5 pp), with smaller positive gains elsewhere (Figure 4, Appendix D.1). Thus, experience-source selection matters alongside the success–failure composition studied in Section 5.1.

Behavioral impact of skill consumption. To understand why skill consumption helps some targets but hurts others, we examine trajectories from contrasting skill configurations in the main SpreadsheetBench matrix: GPT-5.4, which improves substantially after consuming skills, and Qwen3.5-9B, which regresses under certain skills. We characterize the observed changes along three axes: **decision-making behavior**, i.e., what solution strategy the model chooses; **exploratory behavior**, i.e., how the agent builds an understanding of the workbook and task environment before acting; and **tool-use behavior**, i.e., how the chosen strategy is instantiated through concrete operations (details in Appendix D.2). Across both targets, skill consumption *reshapes the default policy* rather than triggering new explicit skill calls: it steers GPT-5.4 toward evaluator-aligned computation and verification, while pushing Qwen3.5-9B toward complex workbook-native workflows that gain structural fidelity at the cost of execution robustness.

Finding (Consumption). Given the *same* skill text, per-target gains can differ substantially, with some targets benefiting strongly while others see little effect or even regress. Skill consumption acts by reshaping the target's default policy, so consumption ability is itself a per-target property that bounds achievable gains.

6 From Diagnosis to Intervention: Meta-Skill Guided Extraction

Now we ask whether the Section 5.2 observation—that useful skills encode concrete failure mechanisms and executable remedies—can be operationalized into actionable criteria that improve skill extraction. This is our **RQ3**: whether our empirical findings can be turned into a concrete, drop-in improvement to skill extraction itself.

A naive starting point is to ask an LLM directly for skill-quality criteria. Doing so yields a generic **plausibility rubric**: seven dimensions covering clarity, completeness, conciseness, logical structure, formatting, tone, and generality (full list in Appendix H.5).

Raw and validated rubrics. We design a fully automated rubric-discovery pipeline: GPT-5.4 contrasts high-gap skill pairs, extracts differences favoring the higher-Δ skill, and iteratively merges them into seven candidate dimensions, the **raw rubric**. Using utility measurements from the development split D_{meta} (defined below), we test which raw dimensions predict utility via pairwise evaluation. Each dimension's *better-rate* is the proportion of pairs where the higher-Δ skill receives a more favorable judgment on that dimension. Three dimensions consistently align with utility and form

the **validated rubric**: Failure Mechanism Encoding, Actionable Specificity, and High-Risk Action Blacklist. The dimension-selection diagnostics and complete rubric definitions are in Appendix H.

Meta-skill guided extraction. We operationalize the validated rubric as a compact **meta-skill**: a generation-time prior inserted into the extractor's system prompt. We compare it against (i) the original (un-guided) extractor prompt and (ii) the same prompt augmented with the plausibility rubric. All three conditions use the same extraction pipeline.

Held-out evaluation. To separate rubric development from final evaluation, each domain is divided into disjoint D_{train}, D_{meta}, and D_{heldout} subsets in a 1:1:1 ratio. The training split supplies experience for skill extraction, the development split supplies utility measurements for deriving and selecting the rubric, and the held-out split is used only for final evaluation. GPT-5.4-mini is the extractor; GPT-5.4, Gemini-3.1-Pro, and Qwen3.5-35B are the targets.

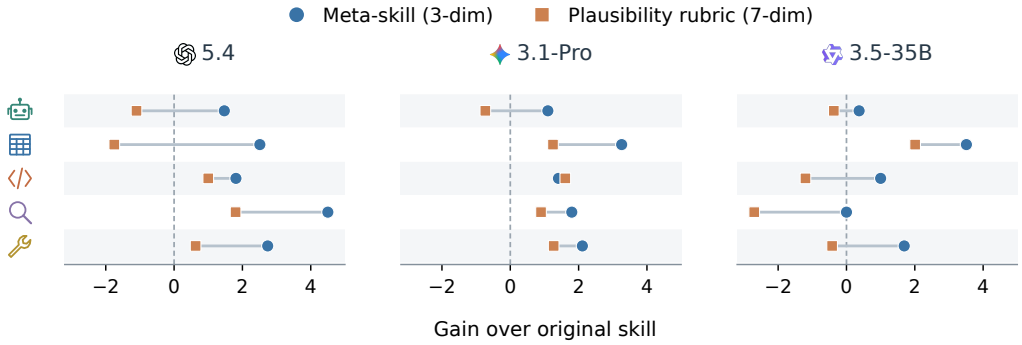


Figure 3: **Effect of meta-skill guidance on held-out skill utility.** Panels are targets and rows are domains (icons as in Figure 2). Connected points compare the validated rubric (blue circles) with the plausibility rubric (orange squares). Gains are in percentage points relative to the original skill (dashed zero line); all values are in Table 12.

Results. As shown in Figure 3, the validated rubric improves 14 of 15 configurations and ties in one, averaging $+1.95$ pp over the original skill; the plausibility rubric averages $+0.15$ pp. The validated rubric also outperforms the plausibility rubric in 14 of 15 configurations. These gains support a drop-in extraction improvement. Cross-domain tests with GPT-5.4-mini as both extractor and target also yield positive gains; full results are in Appendix H.

This closed-loop signal, from diagnostic analysis through dimension validation to measurable downstream improvement, shows that a utility-grounded benchmark can inform not only the evaluation of skills but also the design of skill extraction systems themselves.

Finding (RQ3). We identify a small set of validated rubric dimensions, and using this rubric as a meta-skill to guide the extractor improves generated-skill utility in 14 of 15 held-out configurations, with one tie—demonstrating that empirical findings from this study translate directly into a drop-in extraction improvement.

7 Conclusion

We present a systematic, utility-grounded study of model-generated agent skills across the full lifecycle of experience generation, skill extraction, and skill consumption, spanning five diverse domains and multiple extractors and targets. We find that such skills are beneficial on average but exhibit substantial variance and non-trivial negative transfer, and that neither model scale nor textual plausibility reliably predicts downstream utility. A deep analysis of all three stages, experience generation, skill extraction, and skill consumption, explains where this variance comes from, and we translate these findings into a meta-skill prior, distilled from a validated utility-grounded rubric, which improves extraction in 14 of 15 held-out configurations, with one tie, and plugs directly into the extractor's system prompt. Together, these contributions move agent skill extraction from a heuristic, intuition-driven practice toward a principled, utility-grounded discipline.

Acknowledgments and Disclosure of Funding

This work was supported by Microsoft through Azure API access and computing resources. The authors declare no competing interests.

References

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

[2] Junyu Luo, Weizhi Zhang, Ye Yuan, Yusheng Zhao, Junwei Yang, Yiyang Gu, Bohan Wu, Binqi Chen, Ziyue Qiao, Qingqing Long, et al. Large language model agent: A survey on methodology, applications and challenges. *arXiv preprint arXiv:2503.21460*, 2025.

[3] Anthropic. Claude Skills. https://claude.com/blog/skills, October 2025. Accessed: 2026-05-07.

[4] Libin Qiu, Zhirong Gao, Junfu Chen, Yuhang Ye, Weizhi Huang, Xiaobo Xue, Wenkai Qiu, and Shuo Tang. Autorefine: From trajectories to reusable expertise for continual llm agent refinement. *arXiv preprint arXiv:2601.22758*, 2026.

[5] Dasheng Bi, Yubin Hu, and Mohammed N Nasir. Real-time procedural learning from experience for ai agents. *arXiv preprint arXiv:2511.22074*, 2025.

[6] Qirui Mi, Zhijian Ma, Mengyue Yang, Haoxuan Li, Yisen Wang, Haifeng Zhang, and Jun Wang. Procmem: Learning reusable procedural memory from experience via non-parametric ppo for llm agents. *arXiv preprint arXiv:2602.01869*, 2026.

[7] Salaheddin Alzubi, Noah Provenzano, Jaydon Bingham, Weiyuan Chen, and Tu Vu. Evoskill: Automated skill discovery for multi-agent systems. *arXiv preprint arXiv:2603.02766*, 2026.

[8] Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, et al. Skillrl: Evolving agents via recursive skill-augmented reinforcement learning. *arXiv preprint arXiv:2602.08234*, 2026.

[9] Jingwei Ni, Yihao Liu, Xinpeng Liu, Yutao Sun, Mengyu Zhou, Pengyu Cheng, Dexin Wang, Xiaoxi Jiang, and Guanjun Jiang. Trace2skill: Distill trajectory-local lessons into transferable agent skills. *arXiv preprint arXiv:2603.25158*, 2026.

[10] Hanrong Zhang, Shicheng Fan, Henry Peng Zou, Yankai Chen, Zhenting Wang, Jiayu Zhou, Chengze Li, Wei-Chieh Huang, Yifei Yao, Kening Zheng, Xue Liu, Xiaoxiao Li, and Philip S. Yu. Coevoskills: Self-evolving agent skills via co-evolutionary verification, 2026. URL https://arxiv.org/abs/2604.01687.

[11] Xiangyi Li, Wenbo Chen, Yimin Liu, Shenghan Zheng, Xiaokun Chen, Yifeng He, Yubo Li, Bingran You, Haotian Shen, Jiankai Sun, et al. Skillsbench: Benchmarking how well agent skills work across diverse tasks. *arXiv preprint arXiv:2602.12670*, 2026.

[12] Tingxu Han, Yi Zhang, Wei Song, Chunrong Fang, Zhenyu Chen, Youcheng Sun, and Lijie Hu. Swe-skills-bench: Do agent skills actually help in real-world software engineering? *arXiv preprint arXiv:2603.15401*, 2026.

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

[14] Shiqi Chen, Jingze Gai, Ruochen Zhou, Jinghan Zhang, Tongyao Zhu, Junlong Li, Kangrui Wang, Zihan Wang, Zhengyu Chen, Klara Kaleb, et al. Skillcraft: Can llm agents learn to use tools skillfully? *arXiv preprint arXiv:2603.00718*, 2026.

[15] Yexiong Lin, Shanshan Ye, Yu Yao, Zhen Fang, Bo Han, and Tongliang Liu. SquidAgent: Parallelize wisely, coordinate efficiently. *arXiv preprint arXiv:2610.08647*, 2026. URL https://arxiv.org/abs/2610.08647.

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

[17] Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. *arXiv preprint arXiv:2308.10144*, 2023. URL https://arxiv.org/abs/2308.10144.

[18] Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024. URL https://arxiv.org/abs/2409.07429.

[19] Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. Memp: Exploring agent procedural memory. *arXiv preprint arXiv:2508.06433*, 2025.

[20] Yifan Yang, Ziyang Gong, Weiquan Huang, Qihao Yang, Ziwei Zhou, Zisu Huang, Yan Li, Xuemei Gao, Qi Dai, Bei Liu, Kai Qiu, Yuqing Yang, Dongdong Chen, Xue Yang, and Chong Luo. SkillOpt: Executive strategy for self-evolving agent skills. *arXiv preprint arXiv:2605.23904*, 2026. URL https://arxiv.org/abs/2605.23904.

[21] Jiongxiao Wang, Qiaojing Yan, Yawei Wang, Yijun Tian, Soumya Smruti Mishra, Zhichao Xu, Megha Gandhi, Panpan Xu, and Lin Lee Cheong. Reinforcement learning for self-improving agent with skill library. *arXiv preprint arXiv:2512.17102*, 2025.

[22] Rong Wu, Xiaoman Wang, Jianbiao Mei, Pinlong Cai, Daocheng Fu, Cheng Yang, Licheng Wen, Xuemeng Yang, Yufan Shen, Yuxin Wang, et al. Evolver: Self-evolving llm agents through an experience-driven lifecycle. *arXiv preprint arXiv:2510.16079*, 2025.

[23] Yijia Fan, Zonglin Di, Zimo Wen, Yifan Yang, Mingxi Cheng, Qi Dai, Bei Liu, Kai Qiu, Yue Dong, Ji Li, and Chong Luo. RESOURCE2SKILL: Distilling executable agent skills from human-created multimodal resources. *arXiv preprint arXiv:2606.29538*, 2026. URL https://arxiv.org/abs/2606.29538.

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

[25] Fangzhou Li, Pagkratios Tagkopoulos, and Ilias Tagkopoulos. Skillflow: Scalable and efficient agent skill retrieval system. *arXiv preprint arXiv:2504.06188*, 2025. URL https://arxiv.org/abs/2504.06188.

[26] Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Cote, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. {ALFW}orld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=0IOX0YcCdTn.

[27] Zeyao Ma, Bohan Zhang, Jing Zhang, Jifan Yu, Xiaokang Zhang, Xiaohan Zhang, Sijia Luo, Xi Wang, and Jie Tang. Spreadsheetbench: Towards challenging real world spreadsheet manipulation. *Advances in Neural Information Processing Systems*, 37:94871–94908, 2024.

[28] 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 *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

[29] Thinh Pham, Nguyen Phan Nguyen, Pratibha Zunjare, Weiyuan Chen, Yu-Min Tseng, and Tu Vu. SealQA: Raising the bar for reasoning in search-augmented language models. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=zWb7ueH16c.

[30] Shishir G. Patil, Huanzhi Mao, Charlie Cheng-Jie Ji, Fanjia Yan, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*, 2025.

[31] OpenAI. Introducing GPT-5.4, March 2026. URL https://openai.com/index/introducing-gpt-5-4/.

[32] Google DeepMind. Gemini 3.1 Pro model card, February 2026. URL https://deepmind.google/models/model-cards/gemini-3-1-pro/.

[33] Google DeepMind. Gemini 3.1 Flash-Lite model card, March 2026. URL https://deepmind.google/models/model-cards/gemini-3-1-flash-lite/.

[34] Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

[35] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

A Limitations, Future Work, and Broader Impact

Limitations and future work. Our experimental design intentionally favors interpretability over coverage. In the main matrix, we consolidate each target's experience into a single domain-level skill and supply it directly through the system prompt at evaluation time, so that the observed performance change can be attributed as cleanly as possible to the skill itself rather than to retrieval policies, agentic scaffolding, or other confounding components in the pipeline. This minimal setup enables controlled cross-extractor and cross-target comparisons. We also evaluate an interactive harness and libraries of up to ten skills (Appendix G). Larger repositories, iterative editing, executable composition, stale skills, and evolving environments remain outside these experiments.

Broader impact. Skill libraries built by language agents are increasingly reused across models and deployments, and our study has two practical implications. On the positive side, the utility-grounded evaluation, the validated rubric, and the resulting meta-skill prior give practitioners a concrete way to screen out skills that look fluent but transfer poorly, reducing the chance of silently shipping skills that degrade performance and saving the compute that would otherwise be spent on unhelpful or harmful skill reuse. On the negative side, more effective skill extraction inherits the general risks of methods that make language agents more capable: skills that raise task success can be repurposed for misuse, and skills extracted from imperfect experience pools may carry over biases or unsafe shortcuts from those traces. Mitigating these risks is part of the future work outlined above, in particular evaluating skill safety in richer agentic harnesses and at larger library scales.

B Additional Experimental Details

B.1 Experience Pool Collection

For each (target, domain) pair, we run the target model on the training split for multiple rounds, collecting both successful and failed trajectories. Pool sizes vary across domains, reflecting differences in training-split size and per-task cost.

B.2 Evaluation Details

API access. For all OpenAI GPT and Google Gemini models in our experiments, we set the reasoning effort (`reasoning_effort` for GPT, `thinking_level` for Gemini) to `medium`. GPT models are accessed via the Azure OpenAI API, and Gemini models via the official Google Gemini API.

Data splits. The main experiment generally uses 1:1 experience and test splits. ALFWorld uses official valid_seen/valid_unseen splits, while BFCL-v4 uses the multi-turn subset after filtering instances that trigger provider safety filters. Per-domain counts are reported in Table 11; the separate three-way meta-skill protocol is described in Section 6. Within a domain, all (extractor, target) combinations use the same task splits, so comparisons are not confounded by different test instances.

Repeated runs and aggregation. All entries in Table 1, including the *Base* column, are averaged over three independent evaluation runs.

B.3 Extraction Prompt Templates

This subsection reproduces the prompts that instantiate the two-stage extraction framework of Section 3.2: **per-trajectory analysis**, which converts each trajectory into a pattern set of success and failure patterns; **hierarchical consolidation**, which merges pattern sets level by level into a single consolidated pattern set; and a final **skill synthesis** step, which turns the consolidated pattern set into a schema-conformant skill set via tool calls. Each phase is a single prompted call to E.

Per-trajectory analysis prompt. For each trajectory, the extractor is prompted to extract up to K success patterns (if the trajectory succeeded) or failure patterns (if it failed). The two cases share the same template (Table 2), swapping only the per-type guidance block.

The accompanying user message contains the trajectory's outcome and reward together with the agent's full step-by-step trace; for interactive environments such as ALFWorld we render compact

Table 2: Per-trajectory analysis prompt: extracts a pattern set from one trajectory.

You analyse a single agent trajectory and extract *[success patterns | failure patterns]* – high-level, reusable, transferable behavioural patterns. Focus on genuinely novel, reusable patterns from THIS trajectory; do NOT try to be exhaustive.

What is a pattern? A pattern is a high-level behaviour that is (1) transferable across a broad class of tasks, (2) actionable enough to follow, (3) non-obvious – going beyond common sense, and (4) self-contained – understandable without the original trajectory.

Per-type guidance (success). Capture effective strategies, decision patterns, and methodological insights. Ask: "What did this agent do RIGHT that other agents facing similar tasks should also do?"

Per-type guidance (failure). Capture error patterns, anti-patterns, and non-obvious pitfalls. Ask: "What should an agent AVOID doing when facing similar tasks?"

Quality requirements. Each pattern must be (i) high-level and domain-general, (ii) maximally broad in coverage, (iii) information-dense with a concrete description, and (iv) free of task-specific details (no specific file names, identifiers, error messages, or API calls).

Constraints. Extract at most *[K]* patterns from this trajectory; each as a pattern name and a 2–4 sentence description.

Output format. A JSON list of {type, pattern, description} entries plus a brief summary. If no useful patterns are found, return an empty list.

`[think]/[action]/[obs]` tuples to avoid header duplication, and for trajectories with a free-form final answer the final answer is appended.

Hierarchical consolidation prompt. Pattern sets are merged in groups of G at each level of Equation (2) until a single consolidated pattern set remains. Table 3 shows the merge prompt used at every level.

Skill synthesis prompt. Once a single consolidated pattern set is obtained, the extractor converts it into the skill set via structured tool-calling operations against a writable store (creation, update, and deletion of skills with schema validation). The system prompt shown in Table 4 specifies how patterns are turned into schema-conformant skills.

Optional meta-skill guidance. For meta-skill-guided runs (Section 6), an additional *Extraction Quality Guidance* block — the validated 3-dimension rubric or the 7-dimension plausibility rubric — is appended to the per-trajectory and skill-synthesis prompts.

B.4 Injection Template

At evaluation time, the extracted skill set is exposed to the target model in one of two ways depending on its size.

Single-skill protocol. When there is exactly one skill, we skip the tool protocol and inline the skill body directly into the target's system prompt, using the template in Table 5.

Multi-skill protocol. When the skill library contains multiple skills, the target consumes them through progressive disclosure: it first calls `list_skills` to see names and descriptions, then `view_skill` for the full body, and finally `read_skill_file` for any attached references or scripts. For SpreadsheetBench (which runs in a plain-text conversation rather than via OpenAI function calling), these calls are issued as fenced ```` ```skill ... ``` ```` blocks, analogous to ```` ```python ... ``` ```` code execution blocks. The corresponding system-prompt section is shown in Table 6.

Table 3: Hierarchical consolidation prompt: merges G pattern sets into one.

You receive several pattern sets, each extracted from a different agent trajectory, and merge them into a single consolidated pattern set.

Guidelines.
1. **Deduplicate:** if multiple patterns describe the same or overlapping behaviour, combine them into ONE stronger pattern with the best description.
2. **Generalise:** raise the abstraction level to cover more scenarios; a single well-generalised pattern is worth more than several narrow ones.
3. **Preserve type:** keep success and failure patterns separate; do NOT convert between types.
4. **Preserve quality:** drop vague or low-value patterns; keep concrete, actionable ones.
5. **Prioritise:** when there are too many patterns, retain the most important and broadly applicable ones.

Quality requirements. Each merged pattern must be transferable across tasks, information-dense, non-obvious, and free of task-specific details.

Output format. A JSON object with the consolidated success and failure patterns plus a brief summary of merge decisions.

Table 4: Skill synthesis prompt: converts the consolidated pattern set into a schema-conformant skill set via tool calls.

You receive a consolidated set of success and failure patterns and synthesise them into skills by issuing tool calls against the skill store.

Synthesis strategy.
1. **Integrate both polarities:** a good skill includes both what TO DO (from success patterns) and what to AVOID (from failure patterns).
2. **Organise thematically:** group related patterns into coherent skills around shared themes.
3. **Structure the body clearly:** recommended approaches, common pitfalls, decision criteria for when to apply, and verification methods.
4. **Maintain information density:** every sentence carries actionable content; no platitudes.
5. **Keep the description short:** 1–2 sentences only; all detail goes in the body.

Schema requirements. name (lowercase-hyphen slug, $\leq$ 64 chars); description (1–2 sentences: what class of problems, when to apply); body (Markdown with strategies, pitfalls, decision criteria, verification); optional references and scripts.

Budget. Maximum *[max_skills]* skills, each $\leq$ *[max_skill_chars]* characters (strictly enforced); optional total budget *[max_total_chars]*. On a length error, shorten and retry.

Operating rules. Skills MUST be submitted via the creation tool (plain text in the response is ignored). After all skills are added, the extractor signals completion via the finish tool. On tool errors, the issue is fixed and the call is retried.

Table 5: Single-skill injection prompt template.

```
## Skill Reference

Below is a reusable procedural skill extracted from previous
successful problem-solving experiences.  It may help you solve the
current task more effectively.  Use it as a reference - adapt it to
the specific task at hand.

### [skill.name]

[skill.description]

[skill.body]

#### Reference Files (optional)
[filename]:  [content] ...

#### Script Files (optional)
[filename]:  [content] ...

Note:  This skill is an optional aid, not a mandatory procedure.  Use
your own judgment.
```

Table 6: Multi-skill injection prompt template (text-mode skill tool protocol).

```
## Skill Library

You have access to a Skill Library containing [N] reusable procedural
skills extracted from previous successful problem-solving experiences.
These skills may help you solve the current task more effectively.

Available skill tools.
- list_skills:  see all available skills (name, description).
- view_skill <skill_name>:  read the full body of a specific skill;
also lists attached file names.
- read_skill_file <skill_name> <filename>:  read the content of an
attached reference or script.

How to use skill tools.  Call a skill tool with a ```skill ... ```
block (NOT a ```python ... ``` block).

Workflow.
1.  At the start, call list_skills to see what is available.
2.  If a skill seems relevant, call view_skill to read its body.
3.  If it has attached files, use read_skill_file.
4.  Adapt the skill's guidance to the specific task.
5.  After consulting skills, proceed with ```python ... ``` code
blocks.

Important notes.  Skill tools are read-only.  Each response should
contain EITHER a "```skill"``` block OR a "```python"``` block, not both.
Skills are optional aids, not mandatory procedures.
```

B.5 Extraction Hyperparameters

All extraction experiments use the unified extraction framework with the default settings listed in Table 7 unless otherwise noted:

Table 7: Default extraction hyperparameters.

Parameter	Value
Max patterns per trajectory	3
Merge group size	10
Max skill characters	3000
Max skills per extraction	1
Temperature	1.0

In the *map* phase, each trajectory is independently analyzed to extract up to 3 success or failure patterns. In the *reduce* phase, patterns are grouped in batches of 10 and iteratively merged into a single consolidated skill set. The extractor model, experience pool, and target model vary across experimental conditions as specified in the main text.

B.6 Compute Resource

Closed-source models (GPT-5.4, GPT-5.4-mini, Gemini-3.1-Pro, Gemini-3.1-FL) are accessed through their respective providers' APIs. Open-source models (Qwen3.5-35B, Qwen3.5-9B) are served locally with vLLM [35] on a single node equipped with 8 NVIDIA B200 GPUs, which is sufficient to run all open-source extractors and targets used in the study at the inference scales reported.

C Format Normalization Experiment

We test whether skill utility depends on output format by rewriting the strongest extractor's skill on SpreadsheetBench into four canonical formats: *ordered list* (flat numbered steps), *unordered list* (bullet points), *checklist* (checkbox items), and *prose* (flowing paragraphs). Each rewrite is generated by GPT-5.4 with an instruction to preserve all semantic content while converting to the target format; a verification pass confirms content preservation and format compliance. Each format is evaluated on the same test set as in Section 4 for 3 independent rounds. The four model configurations for this control are listed in Table 8.

Statistical test. We use the Friedman test, a non-parametric repeated-measures analysis of variance. For each target model, the test treats each task instance as a block and each format (or extractor) as a treatment, testing the null hypothesis that all treatments produce equal performance. To quantify effect size relative to noise, we compute σ-ratio $= \sigma_{\text{factor}}/\sigma_{\text{round}}$, where σ_{factor} is the standard deviation of mean performance across factor levels (formats or extractors) and σ_{round} is the standard deviation across independent evaluation rounds with the same factor level. A σ-ratio >1 indicates the factor effect exceeds run-to-run sampling noise.

Table 8: Format vs. extractor effect on SpreadsheetBench. σ-ratio $= \sigma_{\text{factor}}/\sigma_{\text{round}}$; values >1 indicate the factor exceeds noise.

	Format		Extractor	
Target	p	σ-r	p	σ-r
5.4	.983	0.11	<.001	**1.98**
5.4-mini	.916	0.15	<.001	**1.31**
3.5-35B	.345	0.67	<.001	**2.54**
3.5-9B	.452	0.53	.515	0.83

Format has no detectable effect on any target (all $p > 0.34$, all σ-ratios below 1). In contrast, the extractor control yields significant effects for 3/4 targets ($p < 0.005$) with σ-ratios well above 1.

D Experience-Pool Quality and Skill-Consumption Behavior

D.1 Experience-Pool Source Comparison

This experiment supports the experience-pool analysis in Section 5.3. GPT-5.4-mini extracts two SpreadsheetBench skills, one from GPT-5.4 experience and one from Qwen3.5-9B experience, and each is applied to all six consumers. The labels *strong-pool* and *weak-pool* refer to the source models' baseline task performance, rather than to an independently measured quality score for the pools. Unlike the success-ratio manipulation in Section 5.1, this comparison changes the source model and its resulting trajectories.

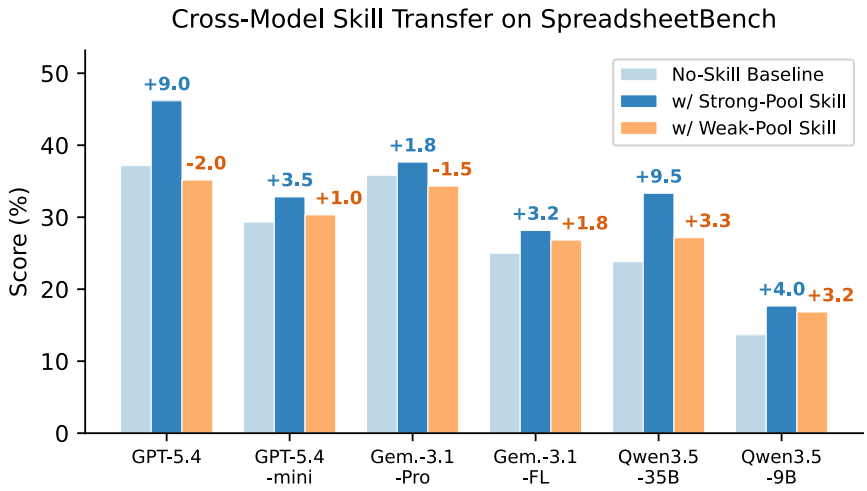


Figure 4: Experience-pool source comparison on SpreadsheetBench. With GPT-5.4-mini fixed as extractor, skills from GPT-5.4 (strong-pool) and Qwen3.5-9B (weak-pool) experience are each applied to six targets. Bars include the no-skill baseline; annotations show gains in pp.

The strong-pool skill improves all six consumers. The weak-pool skill yields negative transfer on GPT-5.4 (-2.0 pp) and Gemini-3.1-Pro (-1.5 pp), and smaller positive gains on the other consumers. Thus, the comparison concerns which experience pool supplies the skill, complementing the fixed-source consumer comparison in Figure 2(b).

D.2 Behavioral Impact Analysis

To better understand why the same intervention helps some models but hurts others, we take a closer look at model behavior on SpreadsheetBench. We focus on two representative cases: GPT-5.4, which improves clearly after consuming skills, and Qwen3.5-9B, which regresses under some consumed skills. We describe the behavior changes from three angles: **decision-making behavior**, **exploratory behavior**, and **tool-use behavior**.

Decision-making behavior. The main change in decision-making is that skill consumption changes how the model frames the task at the beginning. For GPT-5.4, the consumed skill often moves the model away from writing spreadsheet formulas as the final answer and toward computing the result in Python and writing back the final value. This is especially helpful for cell-level tasks, where formula-based answers may look reasonable but are not always stable under evaluation. In other words, skill consumption mostly works as a strategy correction for GPT-5.4: it does not give the model a new ability, but makes it choose a more reliable solution more often.

For Qwen3.5-9B, the shift is less consistently helpful. After consuming a skill, the model is more likely to leave simple dataframe-style heuristics and follow a workbook-native workflow. This can improve structural correctness, especially on sheet-level tasks, because the model is less likely to overwrite the workbook in a crude way. But this also makes the solution process more complex. On fine-grained tasks, the model more easily makes execution mistakes, so the gain in structure sometimes comes with a drop in robustness.

Exploratory behavior. Skill consumption also changes what the model does before editing the workbook. In both models, we more often see early inspection of sheet structure, headers, used ranges, anchors, and target areas. So the effect is not just on the final action; it also changes how the model builds understanding of the workbook first.

For GPT-5.4, this change is usually small but useful. The model becomes a bit less likely to rely on guesses about layout and a bit more likely to ground its edits in the actual workbook structure. For Qwen3.5-9B, the change is stronger. The model more often inspects the workbook before acting, but this extra exploration does not always lead to better execution. In some failure cases, it leads to longer and more complicated reasoning, while the final result is still wrong.

Tool-use behavior. The clearest change in tool use is not that models start making new explicit skill calls. Instead, the consumed skill is usually absorbed into the prompt and changes how the existing tools are used.

For GPT-5.4, the toolset itself stays mostly the same, but the usage becomes more grounded. We more often see bounded write-back, anchor-based addressing, and simple checks after writing. For Qwen3.5-9B, the change is larger. The model shifts from `pandas`-style round-trip rewriting to more `openpyxl`-based in-place editing. This helps preserve workbook structure, but it also creates more chances to fail when the model cannot reliably carry out the more complex workflow.

E Skill-Judge Evaluation

E.1 Exploratory Skill-Pair Diagnostics

This diagnostic setting selects skill pairs by their measured utility gaps. It is separate from the three-way held-out protocol in Section 6; its pair selection and accuracy are computed within the diagnostic evaluation.

A GPT-5.4 judge compares two skills extracted within the same $(M, \mathcal{D})$ group, sees only the two skill texts, and selects the one it expects to yield higher downstream performance. The rubric-free diagnostic uses 9 independent votes per pair (majority vote). The skill presentation order is randomized per pair to mitigate position bias. The full judge prompt is shown in Table 9.

Table 9: Unguided pairwise judge prompt template.

```
You are comparing two agent skill documents meant to help an AI agent.
Domain: [domain description]

Skill 1:
```
[skill text]
```

Skill 2:
```
[skill text]
```

Which skill document will produce better agent performance? You MUST
choose one.
Reply with JSON: {"choice": "Skill 1" or "Skill 2"}
```

The domain description provides a one-sentence characterization of the task environment (e.g., "SpreadsheetBench: the agent writes Python code to manipulate Excel files and produce correct values in specified output cells"). No evaluation rubric or quality criteria are provided, forcing the judge to rely on its own implicit notion of skill quality.

We construct 151 within-group pairs by enumerating all extractor pairs that share the same (target, domain) and whose $\delta = |\Delta_A - \Delta_B|$ exceeds 0.5 pp. This threshold excludes near-ties where the ground-truth ranking is unreliable due to evaluation noise.

E.2 Judge Models and Held-Out Domain Evaluation

Table 10 compares GPT-5.4, Gemini-3.1-Pro, and Qwen3.5-35B on the same rubric-free diagnostic. Gemini-3.1-Pro reproduces the decline in accuracy as the gap grows; Qwen3.5-35B has lower accuracy at large gaps than over all pairs.

Table 10: Rubric-free selection accuracy (%) in the skill-pair diagnostic setting. δ is the absolute utility gap in pp. Bold marks column maxima.

Judge	All	$\delta \geq 2$	$\delta \geq 3$	$\delta \geq 5$
5.4	46	37	24	16
3.1-Pro	48	40	28	21
3.5-35B	**52**	**41**	**42**	**36**

Figure 5 instead evaluates domain-specific pairs on D_{heldout} under the separate three-way protocol. Rubric guidance improves each reported domain/threshold entry.

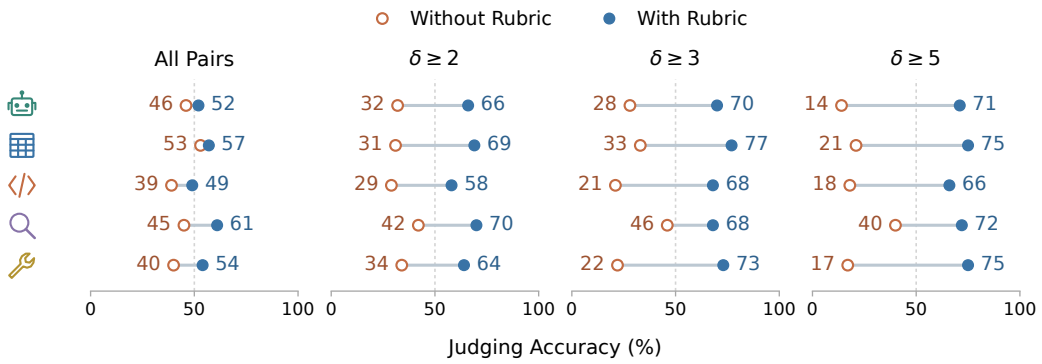


Figure 5: Held-out pairwise judging under the three-way protocol. Open and filled dots show accuracy without and with the utility-grounded rubric; connectors join paired results. Dashed lines mark 50%. Columns stratify by utility gap δ (pp); domain icons follow Figure 2.

F Experimental Scale and Statistical Details

Table 11: Task and trajectory accounting for the main matrix. Trajectories are training tasks $\times$ three runs per target; the last row sums domains for one target.

Domain	Experience	Test	Trajectories / Target
ALFWorld	140	134	420
SpreadsheetBench	200	200	600
SWE-bench-Verified	250	250	750
SEAL-0	55	56	165
BFCL-v4	238	238	714
Total	883	878	2,649

The main experiment contains 150 extracted skills, 450 skill-augmented evaluation runs, and 90 no-skill baseline runs. A run evaluates a complete test split. The trajectory counts in Table 11 are per target and include both successful and failed executions.

Uncertainty and significance. Each cell of Table 1 reports the mean gain and standard deviation across three inference runs for a fixed extracted skill. For the paired test, skill-augmented and no-skill instance scores are first averaged over the three runs, then paired differences are tested across the test instances. Stars denote unadjusted $p < 0.05$. Of the 150 configurations, 114 (76%) have significant

differences; 23 of those are negative (20.2% of the significant configurations). These tests concern inference variability, not repeated skill extraction, and do not constitute a significance test of every extractor ranking.

F.1 Task-Grader Reliability

Four domains use programmatic grading: goal-state checking for ALFWorld, target-cell comparison for SpreadsheetBench, test execution for SWE-bench-Verified, and AST/execution checking for BFCL-v4. SEAL-0 is model-graded using the question, reference answer, and model prediction. We re-grade GPT-5.4-mini's three runs on 56 SEAL-0 test instances, yielding 168 predictions. Three repetitions of the GPT-5.4 grader agree on 167/168 predictions (99.4%). GPT-5.4, GPT-5.4-mini, and Gemini-3.1-Pro agree on 165/168 predictions (98.2%).

G Interactive Harness and Skill Libraries

G.1 Five-Domain Codex Evaluation

To examine interactive consumption, we cross three extractors and three targets—GPT-5.4, GPT-5.4-mini, and Gemini-3.1-Pro—in each of five domains under Codex. All entries compare against the corresponding no-skill Codex baseline. The Codex and skill-library results report mean ± SD over three inference runs. The skill system supports on-demand discovery and selection. Model logos and version labels follow Table 1.

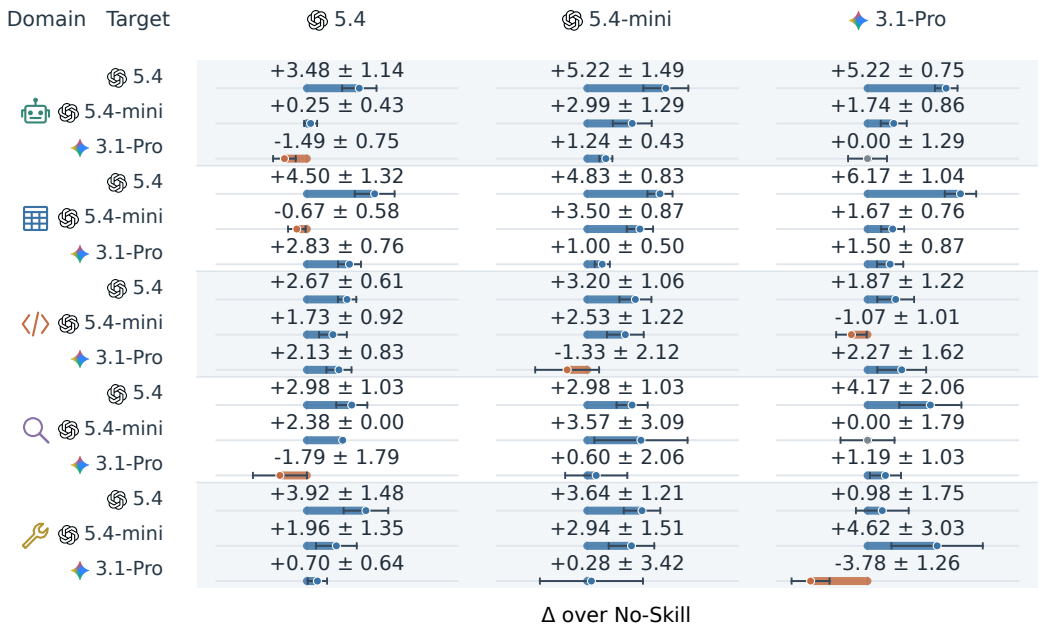


Figure 6: Five-domain Codex evaluation. Rows group targets by domain; columns are extractors. All 45 labels give mean gains ± SD (pp) over three inference runs against the no-skill Codex baseline. Bars and whiskers display the same means and SDs on a shared scale. Domain icons follow Figure 2.

The 45-configuration experiment includes 37 positive, six negative, and two zero deltas. Mean extractor gains across the five domains and three targets are +1.71 pp for GPT-5.4, +2.48 pp for GPT-5.4-mini, and +1.77 pp for Gemini-3.1-Pro.

G.2 Skill-Library Size

In Figure 7, each model is both extractor and consumer, and the number of available skills varies over 1/5/10. Ten skills achieve the highest mean in 11 of 15 model–domain combinations, five in

three, and one in one. For example, Gemini-3.1-Pro on SWE-bench-Verified is best with a single skill, while its BFCL gains remain negative at all three library sizes.

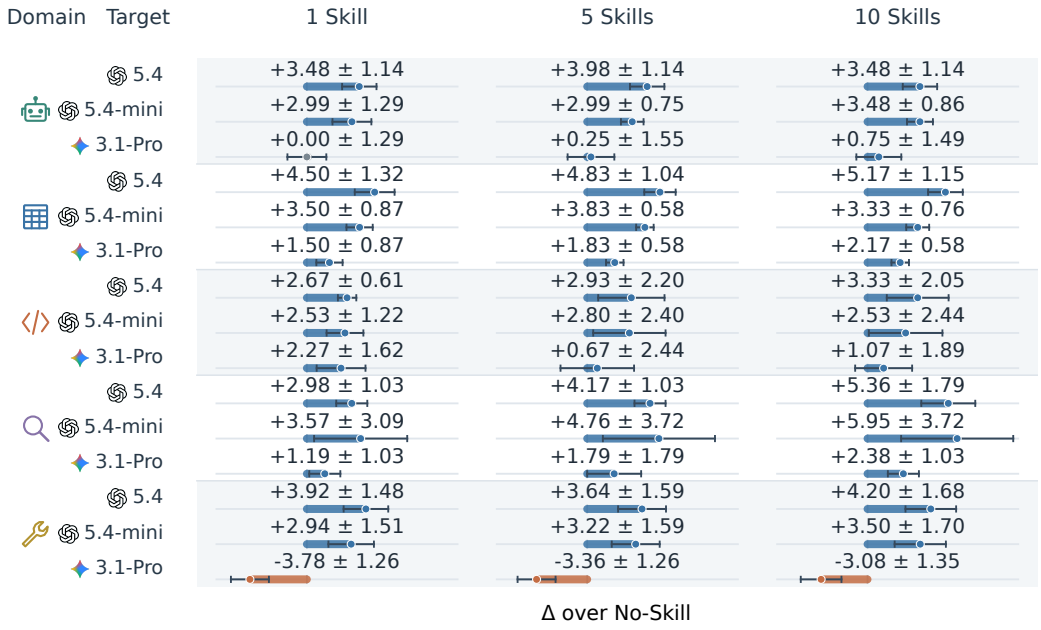


Figure 7: Effect of skill-library size under Codex. Each row uses the same model as extractor and target; columns vary the library over 1/5/10 skills. All 45 labels give mean gains ± SD (pp) over three inference runs against the no-skill Codex baseline. Domain icons follow Figure 2; the gain range matches Figure 6.

H Meta-Skill Analysis

H.1 Held-Out In-Domain Evaluation

The three-way protocol in Section 6 separates experience generation, rubric development, and final evaluation. The in-domain experiment fixes GPT-5.4-mini as extractor and evaluates GPT-5.4, Gemini-3.1-Pro, and Qwen3.5-35B as targets. Table 12 reports the complete per-target increments over the original skill. The overall mean equally weights the 15 configurations. Figure 3 displays every per-target result.

Table 12: Held-out meta-skill evaluation under the three-way split. Entries are gains (pp) over the original skill. GPT-5.4-mini is the extractor; the original skill is the zero reference. Plaus. denotes plausibility guidance; bold marks the larger gain within each target pair.

Domain	5.4		3.1-Pro		3.5-35B	
	Plaus.	Meta	Plaus.	Meta	Plaus.	Meta
ALFWorld	−1.10	**+1.47**	−0.73	**+1.10**	−0.37	**+0.37**
SpreadsheetBench	−1.75	**+2.51**	+1.25	**+3.26**	+2.01	**+3.51**
SWE-bench-Verified	+1.00	**+1.81**	**+1.61**	+1.41	−1.20	**+1.00**
SEAL-0	+1.80	**+4.50**	+0.90	**+1.80**	−2.70	**+0.00**
BFCL-v4	+0.63	**+2.74**	+1.27	**+2.11**	−0.42	**+1.69**

Meta-skill guidance improves 14 configurations and ties in the Qwen3.5-35B/SEAL-0 configuration. Its mean increment is +1.952 pp, compared with +0.147 pp for plausibility guidance. The plausibility rubric performs better in one configuration (Gemini-3.1-Pro on SWE-bench-Verified, +1.61 versus +1.41 pp).

H.2 Cross-Domain Transfer

For cross-domain transfer, a meta-skill is derived from one domain's development data and evaluated on the held-out tasks of the evaluation domain. Both extractor and target are fixed to GPT-5.4-mini. The Mixed row pools development data from all five domains. Every reported increment is positive, with domain-specific and mixed derivation offering different tradeoffs.

Table 13: Cross-domain meta-skill transfer under the three-way split. Rows are derivation domains and columns evaluation domains; Mixed pools all five derivation domains. Values are gains over the original skill (pp); extractor and target are GPT-5.4-mini. The shaded diagonal denotes in-domain derivation; bold marks column maxima (ties included).

Derived on	ALFWorld	SpreadsheetBench	SWE-bench-Verified	SEAL-0	BFCL-v4
ALFWorld	**+0.61**	+1.67	+0.54	+1.20	+1.27
SpreadsheetBench	+0.37	**+3.26**	+0.54	+1.20	+1.41
SWE-bench-Verified	+0.37	+1.92	+0.87	+1.20	+1.27
SEAL-0	+0.37	+1.92	+0.54	**+2.10**	+1.41
BFCL-v4	+0.37	+2.34	+0.60	+1.20	+2.11
Mixed	**+0.61**	+3.01	**+0.94**	+1.80	**+2.32**

H.3 Cross-Family Transfer

For cross-family transfer, derivation excludes all cells whose extractor or target belongs to the held-out family. The derived meta-skill is then evaluated on cells where extractor and target are both members of that family. Gains are averaged within family and domain. The reported family averages are +0.92 pp for GPT and +1.60 pp for Gemini, supporting transfer to these held-out families while showing asymmetry.

Table 14: Leave-one-family-out meta-skill transfer: gains over the original skill (pp). Derivation excludes all cells with an extractor or target from the held-out family. Bold marks column maxima.

Held-Out Family	ALFWorld	SpreadsheetBench	SWE-bench-Verified	SEAL-0	BFCL-v4	Avg.
GPT	**+0.64**	+1.50	+0.25	+1.13	+1.05	+0.92
Gemini	+0.37	**+2.76**	**+1.05**	**+1.80**	**+2.00**	**+1.60**

H.4 Dimension Ablation and a Failure Case

The dimension ablation in Table 15 uses the same three-way split and GPT-5.4-mini as both extractor and target; its Full Meta-Skill condition corresponds to the in-domain diagonal of Table 13. Most individual dimensions help, and the full rubric performs best in each domain. The exception is Actionable Specificity on SEAL-0 (-0.42 pp), illustrating a limitation of prescribing a fixed procedure.

Table 15: Single-dimension guidance and the full meta-skill in a supplementary ablation. Gains are relative to the original skill (pp); extractor and target are GPT-5.4-mini.

Guidance	ALFWorld	SpreadsheetBench	SWE-bench-Verified	SEAL-0	BFCL-v4
Failure Mechanism Encoding	+0.42	+1.85	+0.55	+0.98	+0.87
Actionable Specificity	+0.38	+2.40	+0.48	−0.42	+0.95
High-Risk Action Blacklist	+0.55	+1.42	+0.61	+1.15	+1.02
Full Meta-Skill	**+0.61**	**+3.26**	**+0.87**	**+2.10**	**+2.11**

One skill generated with this single dimension contains the instruction: *"Do not answer from search snippets. Open the official report or data table, locate the exact field, and quote the value from there."* This instruction can correct unreliable snippet-based answers. However, for a question whose disambiguating information appears on an aggregator page while the official table uses a different definition, the procedure can direct the agent to an authoritative but irrelevant value. It shows why domain-level procedures require task-sensitive application.

H.5 Plausibility Rubric (Naive Baseline)

The plausibility rubric is obtained by directly asking GPT-5.4 to enumerate seven quality criteria for agent skills, with no exposure to actual skill pairs or downstream-utility data. By construction, the resulting dimensions describe what an LLM *believes* would distinguish a good skill from a bad one—surface qualities of the text—rather than properties grounded in observed utility. Table 16 lists the seven dimensions used as the plausibility-guidance baseline in Section 6, distinct from extraction without any rubric.

Table 16: Seven dimensions of the **plausibility rubric**, generated directly by GPT-5.4 without any pair-level or utility-level grounding.

#	Dimension	Definition
1	**Clarity**	language is unambiguous and free of confusing phrasing
2	**Completeness**	covers the full scope of typical task scenarios
3	**Conciseness**	dense and free of unnecessary verbosity
4	**Logical Structure**	sections are organized in a coherent order
5	**Formatting Quality**	uses headers, lists, and code blocks where appropriate
6	**Tone Neutrality**	phrasing is professional and free of biased language
7	**Generality**	applies broadly across tasks rather than to a single example

H.6 Raw Rubric (from Contrastive Pipeline)

The contrastive discovery analysis synthesizes seven candidate quality dimensions from recurring themes across 17 high-gap skill pairs. The better-rates below are discovery diagnostics, not the held-out effectiveness measurements in Figure 3. Table 17 lists all seven dimensions of this **raw rubric** with their definitions and per-dimension better-rates (the proportion of pairs where the higher-Δ skill receives more favorable judgments on that dimension).

Table 17: Seven dimensions of the **raw rubric**, discovered via the automated contrastive pipeline. Better-rate measures alignment with downstream utility; the three **bold** dimensions form the **validated rubric** used for guided evaluation and meta-skill extraction.

#	Dimension	Definition	Better-Rate
1	**Failure Mechanism Encoding**	identifies *why* agents fail, not just *that* they fail	**65.5%**
2	**Actionable Specificity**	step-level procedures referencing domain objects/tools	**66.0%**
3	Environment/Tool Semantics	encodes how tools and objects actually behave	63.2%
4	Strategy Switching Conditions	specifies when to change approach	47.5%
5	Boundary Condition Coverage	addresses specific edge cases	63.0%
6	**High-Risk Action Blacklist**	forbids specific harmful action patterns	**64.6%**
7	Benchmark-Aligned Priorities	addresses what evaluation actually measures	56.2%

H.7 Representative Contrastive Cases

Tables 18 and 19 show representative best-vs-worst skill pairs from SpreadsheetBench and ALFWorld. We reproduce the full skill text and highlight key passages: green marks domain-specific failure mechanisms or executable countermeasures; red marks generic advice that provides little actionable leverage.

Table 18: Contrastive case: SpreadsheetBench, target = GPT-5.4, Δ gap = 10.3 pp.

Higher-Δ skill ($\Delta = +14.7$, ext = Gemini-3.1-FL)	**Lower-Δ skill** ($\Delta = +4.3$, ext = GPT-5.4)
Treat spreadsheet files solely as I/O containers. Never rely on the host application to evaluate formulas or perform business logic. **1. Proactive Reconnaissance.** Diagnostic Audit: read all sheets, row counts, headers, sample rows, and merged-cell maps *before* any mutation. Dynamic Addressing: search for anchor data (e.g., column headers) to determine indices; never use hardcoded cell references. Normalization: establish a cleaning layer before processing. **2. In-Memory Processing.** Logic Decoupling: extract data into Python structures; perform all aggregations in memory. Avoid Formula Injection: writing formula strings does not trigger calculation engines in headless environments. Always calculate the final static value in Python and write the scalar result. **3. Idempotent Write Strategy.** Atomic Updates: clear target ranges before writing. Reverse Iteration: when deleting or rearranging data, iterate bottom-to-top to avoid index-shifting errors. Metadata Preservation: use style-preserving libraries. **4. Post-Execution Validation.** Verification Loop: perform a post-write audit to confirm output matches expected logic. Fail-Fast: if an intermediate step fails, simplify rather than patch. **Critical Pitfalls:** Formula Injection Fallacy; Verification Blindness; Destructive Mutation; Context-Agnostic Recycling.	**1.** Inspect the live artifact first. Confirm what you are editing and roughly where the relevant scope is before writing anything. **2.** Resolve the contract before coding. Determine exact deliverable: edited artifact, formulas vs values, write scope, preservation requirements. **3.** Derive logic from semantic anchors. Use headers, labels, markers, nearby formulas; do not rely on fixed coordinates. **4.** Normalize into a canonical model. Trim/case-normalize text, parse compound cells, coerce types safely. **5.** Stage the work. Separate discovery, computation, mutation, and formatting. Prove the core rule on representative cases before bulk changes. **6.** Choose the simplest method that matches the contract and runtime. **7.** Edit minimally and safely. Keep changes inside the intended scope and avoid disturbing unrelated parts of the artifact. **8.** Round-trip validate the saved result. Reopen the artifact and verify target cells, formulas or values. **Pitfalls:** Trusting stale inspection; hardcoding coordinates; guessing ambiguous rules; mixing exploration with mutation; treating successful execution as proof.
Analysis. The higher-Δ skill encodes three *domain-specific failure mechanisms* absent from the lower skill: (1) the formula injection fallacy—formulas are not evaluated in headless execution, so agents must precompute static values; (2) index-shifting errors during deletion, countered by reverse iteration; (3) dynamic addressing to avoid hardcoded coordinates. Each mechanism is paired with an executable remedy. The lower-Δ skill, by contrast, relies on process-level directives ("resolve the contract," "edit minimally") that are reasonable but too abstract to prevent the concrete failure modes that dominate SpreadsheetBench errors.	
Guided rubric judgment (3 validated dimensions)	
Failure Mechanism Encoding	
✓	✗
Actionable Specificity	
✓	✗
High-Risk Action Blacklist	
✓	✗

Table 19: Contrastive case: ALFWorld, target = GPT-5.4, Δ gap = 6.0 pp.

Higher-Δ skill ($\Delta = +7.5$, ext = Gemini-3.1-Pro)	**Lower-Δ skill** ($\Delta = +1.5$, ext = GPT-5.4)
1. Search Strategy & Spatial Memory. Semantic to Systematic: begin searching high-probability locations based on semantics. If not found, transition to an exhaustive sweep of ALL open surfaces and closed receptacles. **Deep Inspection:** never merely observe the exterior of closed receptacles. You MUST explicitly `open` them and inspect contents to avoid false negatives. State and Spatial Memory: maintain a checklist of explored areas to prevent amnesic looping. Memorize incidental item locations for later retrieval. **2. Strict Pipelining.** Linear Execution Pipeline: Locate $\rightarrow$ Acquire $\rightarrow$ Transform $\rightarrow$ Navigate $\rightarrow$ Deposit. Complete each phase before advancing. **Active State Transformations:** if an object requires a state change (cleaned, heated), locate it, acquire it, transport it to the appliance, invoke the command, and verify. Exact Lexical Matching: adhere strictly to the requested target object name; never substitute synonyms. **3. Preconditions & Multi-Item Transport. Proactive Prerequisite Resolution:** verify and resolve physical preconditions (navigating to proximity, opening destination receptacles) *before* attempting core interactions. Incremental Fetch-and-Deliver: for multi-item tasks, use single-item fetch-and-deposit cycles. **Pitfalls:** Redundant state verification; semantic fixation; premature goal reversal.	**1.** Ground the goal exactly. Translate the instruction into explicit predicates and act on them in order. **2.** Find the current bottleneck. Work backward from success and act on the earliest unmet prerequisite. **3.** Search with memory and pivot rules. Start with visible, nearby, semantically likely candidates. Keep a ledger of searched locations, opened objects, confirmed sources, held items, remaining counts. If a location class yields repeated misses, broaden to a new region. **4.** Manage preconditions through affordances. Before key actions, make sure access and usability are in place. Treat failed actions as evidence of a missing prerequisite, not a cue to retry. **5.** Bank monotonic progress. When you find a valid item, convert it into durable progress quickly. For repeated goals, use acquire-deliver-repeat loops. **6.** Replan on observation; finish minimally. After each observation, recheck what is still unsatisfied. Once a valid completion path exists, stop exploring and execute the shortest finish chain. **Failure patterns:** searching without coverage memory; shallow inspection treated as proof; stale-plan repetition; endgame thrashing.
Analysis. The higher-Δ skill provides three *executable action patterns* tailored to ALFWorld's mechanics: (1) deep inspection—explicitly `open` closed containers rather than assuming visibility equals absence; (2) active state transformations—a concrete locate-acquire-transport-invoke pipeline for state changes; (3) prerequisite resolution—navigate and open destinations *before* attempting placement. The lower-Δ skill describes the same high-level logic ("ground the goal," "find the bottleneck," "manage preconditions") but at a level of abstraction that does not map onto ALFWorld's action vocabulary, leaving the agent to rediscover the operational details on its own.	
Guided rubric judgment (3 validated dimensions)	
Failure Mechanism Encoding	
✓	✓
Actionable Specificity	
✓	✓
High-Risk Action Blacklist	
✓	✗

I AI Use Disclosure

Generative language models are experimental components in this study: they execute tasks, extract skills, produce rubric candidates, and judge skill pairs; a language model also grades SEAL-0 answers. AI assistance was used to organize existing experimental material, analyze reported results, prepare text, tables, and figure-generation code, and adapt the LaTeX source to the conference format. The authors are responsible for the final text, claims, and artifacts.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We detailed discuss our main claims and contribution in the abstract and introduction.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discuss the limitation of the work and future plan in Appendix A

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

Justification: We do not have theoretical results.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Yes, we provide all the parameter, details and code for replication.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: Yes, we provide the full code. The dataset we use are open-source resources.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

Answer: [Yes]

Justification: We clearly state the splits ratio, hyperparameters in Appendix.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: We include statistical tests in some of our experiments as a part of analysis.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.

- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We discuss this in Appendix B.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

   Answer: [Yes]

   Justification: Yes, this research conform with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Yes, we discuss this in Appendix A.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: Our study do not have safety risks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We use all research-community open-sourced datasets and only for research purpose. We have cited properly.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, paperswithcode.com/datasets has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We discuss our implemented code and provide code files in attache zip.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: We do not involve research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: We do not involve research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: We study LLMs, so LLMs are used in our main experiments and analysis. We discuss the usage of LLM and give detailed prompt in the corresponding sections.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.