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

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.

Correspondence: yifanyang@microsoft.com, zhengxq@fudan.edu.cn

Code: https://aka.ms/SkillLens

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

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 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 that analyze 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

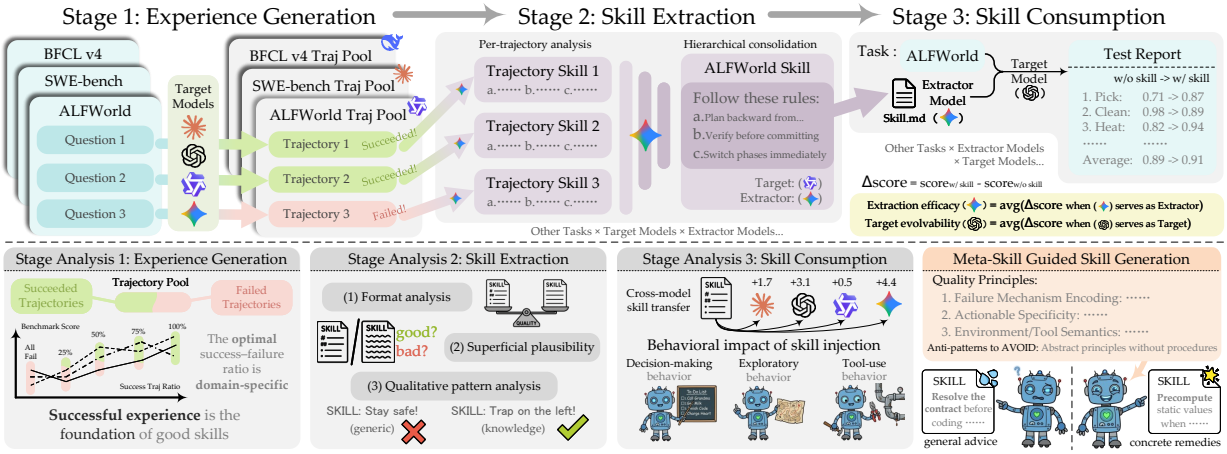

Figure 1 Overview of our study design. We evaluate the full trajectory-to-skill lifecycle across three stages: experience generation, skill extraction, and skill consumption.

stable and effective when deployed in real agent systems.

2 Related Work

Automatic Generation of Reusable Knowledge from Agent Experience. Recent surveys identify agent skills—composable packages of instructions, code, and resources loaded on demand—as a defining mechanism for extending LLM capabilities without retraining [1], 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. *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 patterns, PRAXIS [5] builds state-indexed procedural memory, and MemP [15] 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, and others combine skill banks with reinforcement learning [7, 8, 16]. A third line studies *self-evolving lifecycle agents* that iteratively refine skills through closed-loop deployment, as in EvolveR [17]. 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. 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 Liu et al. [13] primarily test whether curated or discovered skills improve downstream performance over a no-skill baseline. Another emphasizes *retrieval and orchestration at scale*: AgentSkillOS [18] studies ecosystem-level skill management, while SkillFlow [19] 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.

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:

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

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:

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

where Perf is the domain-specific task metric. Baseline and skill-augmented evaluations use the same held-out split $Q_{\mathcal{D}}^{\mathrm{test}}$. $\Delta > 0$ 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 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** [20]: embodied household tasks requiring physical commonsense, exploration, and multi-step planning.
- **SpreadsheetBench** [21]: spreadsheet manipulation tasks involving table inspection, formula

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

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	+6.47	+7.46	+4.98	+4.23	**+4.93**
5.4-mini	52.24	+1.00	+4.23	+2.74	+2.24	+3.98	+2.84
3.1-Pro	87.56	+0.50	+0.75	+0.00	−0.75	−1.24	−0.15
3.1-FL	51.99	−2.49	−1.24	+1.49	−2.49	−3.23	−1.59
3.5-35B	57.21	−1.99	−3.48	−0.75	+0.50	−1.00	−1.34
3.5-9B	36.07	−2.49	−2.99	−1.24	−1.99	+0.25	−1.69
EE		−0.66	+0.62	**+1.62**	+0.42	+0.50	
Productivity: *SpreadsheetBench*							
5.4	37.17	+4.33	+9.00	+14.00	+14.66	+6.33	**+9.66**
5.4-mini	29.33	+0.34	+2.50	+3.67	+4.50	+1.00	+2.40
3.1-Pro	35.83	−0.50	−2.67	+6.50	+5.33	+5.83	+2.90
3.1-FL	25.00	+2.67	+1.83	+1.50	+6.17	+7.33	+3.90
3.5-35B	23.83	+2.00	+5.50	+0.17	+3.34	−3.50	+1.50
3.5-9B	13.67	+1.16	+3.16	−1.17	+1.16	+3.00	+1.46
EE		+1.67	+3.22	+4.11	**+5.86**	+3.33	
Coding: *SWE-bench-Verified*							
5.4	68.40	+4.67	+1.33	+2.00	+4.00	+2.27	+2.85
5.4-mini	59.73	+3.20	+3.20	+1.73	+3.60	+2.80	**+2.91**
3.1-Pro	66.53	+2.00	+2.80	+2.13	+3.47	−1.60	+1.76
3.1-FL	55.47	+2.67	+3.33	+2.93	+3.47	−0.93	+2.29
3.5-35B	52.93	+3.20	+2.00	+2.53	+2.93	+2.00	+2.53
3.5-9B	33.33	−1.07	+2.40	−1.60	+1.20	+0.93	+0.37
EE		+2.45	+2.51	+1.62	**+3.11**	+0.91	
Web Search: *SEAL-0*							
5.4	51.24	+6.47	+4.23	+7.71	+1.74	+1.74	+4.38
5.4-mini	45.27	−1.49	+3.23	−3.98	+3.98	−4.23	−0.50
3.1-Pro	55.97	−4.23	−1.99	+1.99	+2.49	−3.48	−1.04
3.1-FL	14.93	+9.45	+8.21	+2.99	−1.24	+7.21	**+5.32**
3.5-35B	40.55	+1.74	+6.47	−3.73	+4.73	+2.24	+2.29
3.5-9B	33.83	+10.70	+8.96	−5.72	+5.97	−2.99	+3.38
EE		+3.77	**+4.85**	−0.12	+2.95	+0.08	
Tool Calling: *BFCL-v4*							
5.4	51.68	+3.08	+5.04	+0.42	+5.04	−2.24	+2.27
5.4-mini	53.50	+3.92	+6.16	+7.56	+5.18	+2.94	**+5.15**
3.1-Pro	51.82	+5.32	+5.88	−4.34	+0.14	+6.02	+2.60
3.1-FL	41.18	+4.06	+4.62	+4.20	+8.12	+3.64	+4.93
3.5-35B	57.56	−0.70	+0.84	+1.54	+2.80	+1.82	+1.26
3.5-9B	46.78	+2.94	+2.94	+1.82	−0.98	−1.12	+1.12
EE		+3.10	**+4.25**	+1.87	+3.38	+1.84	

Table 1 Skill-induced performance gain (Δ) across domains. *Base* is the no-skill baseline. TE denotes **Target Evolvability**, averaged across extractors, and EE denotes **Extraction Efficacy**, averaged across targets. Green: $\Delta > 0$; Red: $\Delta < 0$.

reasoning, filtering, and value editing.

- **SWE-bench-Verified** [22]: real-world software engineering tasks requiring codebase understanding, fault localization, and patch generation.
- **SEAL-0** [23]: web-search question answering tasks requiring retrieval, evidence synthesis, and multi-hop reasoning.
- **BFCL-v4** [24]: 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) [25], **Gemini** (Gemini-3.1-Pro [26], Gemini-3.1-Flash-Lite [27]), and **Qwen** (Qwen3.5-35B, Qwen3.5-9B) [28]. 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 split task instances 1:1 into a experience-generation split $Q_{\mathcal{D}}^{\text{train}}$ and a held-out test split $Q_{\mathcal{D}}^{\text{test}}$; if an official training split exists, $Q_{\mathcal{D}}^{\text{train}}$ is sampled from it at the same proportion. Each target M runs $Q_{\mathcal{D}}^{\text{train}}$ to form a 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) in percentage points. Full extraction and evaluation details are in Appendix B.

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 75% of entries. Yet negative transfer remains common: 25% of 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.

> **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 on SpreadsheetBench, SWE-bench-Verified, and ALFWorld with three targets and report the average Δ.

Results. Figure 2 shows that **experience composition strongly affects extracted skill quality**. Beyond this, **the optimal success–failure ratio is domain-specific**. SpreadsheetBench favors more successful trajectories, SWE-bench-Verified peaks with a mostly successful pool, and ALFWorld performs best with failure-heavy pools. 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 also shows that all-failure pools consistently perform worst, 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.

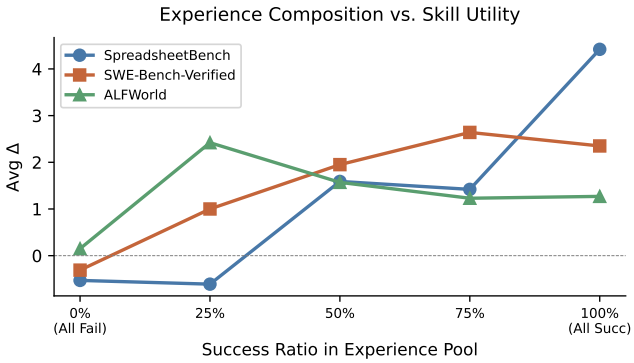

Figure 2 Effect of success ratio in the experience pool on downstream tasks.

> **Finding (Experience).** Skill quality appears to be largely influenced by the success–failure composition of the experience pool. Pure-failure pools are consistently the worst, 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 ask whether shallow textual features of a skill can explain its downstream gains. We rule out two such candidates and surface a qualitative pattern that motivates the systematic analysis in Section 6.

Skill quality is not reducible to surface form. A natural first concern is that skill format may largely influence skill utility. We test this by rewriting the same skill into four canonical formats (*ordered list*, *unordered list*, *checklist*, and *prose*) and re-evaluating each rewrite. We then run a Friedman test, which ranks the four formats within each task and asks whether some format

is consistently ranked higher than the others across tasks. Results in Table 8 (Appendix C) show that the format effect is non-significant on every target (all $p>0.34$), whereas swapping the extractor produces a clearly discernible effect on 5/6 targets ($p<0.01$). This contrast indicates that variance is driven by 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? We probe this with a GPT-5.4 judge as a human proxy. For a pair of skills extracted within the same $(M, \mathcal{D})$, the judge sees only the two skill texts and selects the one it deems higher-quality (better downstream performance). We evaluate on 151 pairs whose $\delta = |\Delta_A - \Delta_B|$ exceeds 0.5%, excluding near-ties (details in Appendix C). Without any evaluation criteria, overall LLM selection accuracy is 46.4%, indistinguishable from random.

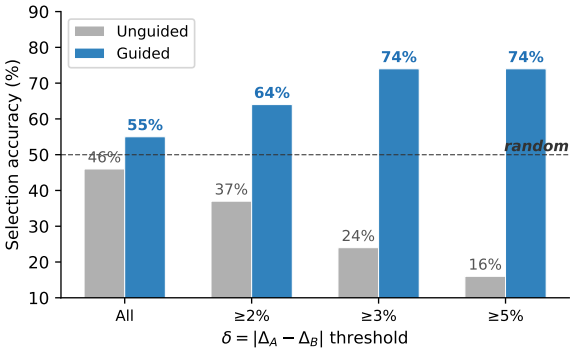

Figure 3 Pairwise selection accuracy by δ.

The gray bars in Figure 3 break this number down by δ: more strikingly, accuracy *decreases* as δ grows. On pairs with $\delta \geq 5\%$, the judge picks the higher-Δ skill only 15.8% of the time, a clear inversion of actual utility. In other words, the skill that reads better is often the one that performs worse. Textual plausibility has come apart from downstream skill utility, a gap we close in Section 6 by identifying which textual properties carry genuine predictive signal.

A qualitative hint: concrete remedies, not generic advice. A qualitative inspection of one high-δ pair in SpreadsheetBench (Table 14, 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 which textual dimensions actually predict utility.

> **Finding (Extraction).** Neither skill format nor textual plausibility predicts utility: directly asking an LLM to judge the skill text performs no better than chance. What truly drives skill utility lies deeper than surface form, 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. Our goal here is to ask how the same skill behaves when consumed by different targets. To sharpen the comparison, we fix a single extractor (GPT-5.4-mini) and select two contrasting skills from its main-experiment outputs on SpreadsheetBench: a *strong-pool skill* distilled from the strongest baseline target's experience pool (GPT-5.4) and a *weak-pool skill* from the weakest (Qwen3.5-9B). These two skills are applied to all six targets. Two patterns emerge in the results shown in Figure 4. First, with the skill text held fixed, per-target gains differ sharply—the strong-pool skill ranges from +1.8 on Gem-3.1-Pro to +9.5 on Qwen3.5-35B, and a similar spread holds for the weak-pool skill—showing that **skill consumption ability varies across targets**. Second, the strong-pool skill consistently improves every target, whereas the weak-pool skill yields clear negative transfer on some targets (e.g., −2.0 on GPT-5.4) and only modest gains

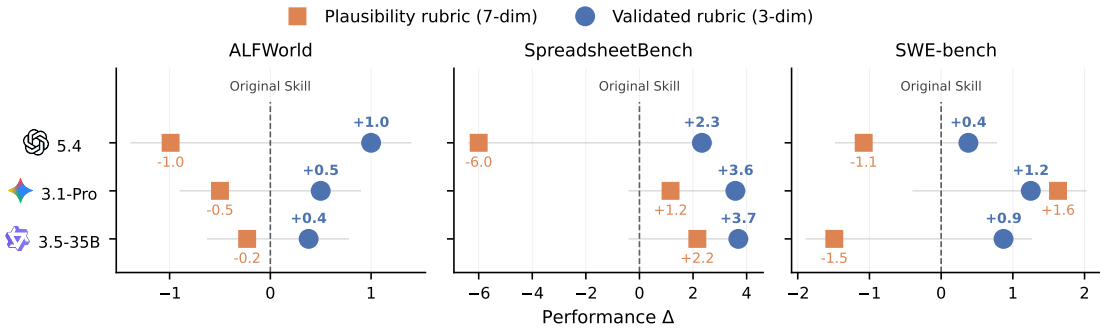

Figure 5 **Effect of meta-skill guidance on downstream skill utility.** The plausibility rubric hurts most times, while the validated rubric improves all the generated skills compared with original skill.

on others—a gap that, in turn, echoes the Section 5.1 finding that **experience-pool quality is critical to the skills it produces**.

Behavioral impact of skill consumption. To understand why skill consumption helps some targets but hurts others, we systematically examine agent trajectories on two contrasting targets on SpreadsheetBench: 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). 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.

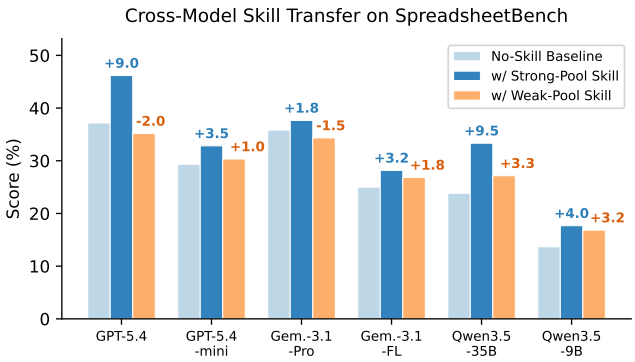

Figure 4 Cross-model skill transfer. Strong-pool and weak-pool skills are injected into each target separately.

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 finding—that textual plausibility does not predict downstream utility—can be operationalized into actionable criteria that improve both skill evaluation and 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.1).

Raw and validated rubrics. We design a fully automated rubric-discovery pipeline that takes the high-gap skill pairs from the cross-matrix as input. GPT-5.4 first analyzes each pair to extract per-pair differences along which the higher-Δ skill outperforms the lower one; these differences are then iteratively merged and consolidated into seven candidate dimensions, which we call the **raw rubric** (Appendix H). We then test which raw dimensions actually predict utility via pairwise evaluation, measuring each dimension's *better-rate*—the proportion of pairs where the higher-Δ skill receives more favorable judgments. Three dimensions consistently align with utility: **Failure Mechanism Encoding**, **Actionable Specificity**, and **High-Risk Action Blacklist** (better-rates 64–66%); together they form the **validated rubric**.

To verify that the validated rubric carries genuine evaluative signal, we feed it back into the same pairwise-judgment protocol from Section 5.2: on the same 151 high-gap pairs, the judge is now instructed to score each candidate along the three validated dimensions and aggregate them into a single preference. This rubric-guided judgment raises overall judge accuracy from 46.4% (unguided) to **73.8%**. The improvement also extends to the hardest pairs ($\delta \geq 5$ pp), where the unguided judge had picked the higher-Δ skill only 15.8% of the time and the guided judge now picks correctly the majority of the time (Figure 3). With the validated rubric, the same LLM judge that previously favored more fluent but worse-performing skills becomes a reliable utility predictor.

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. As shown in Figure 5, the plausibility rubric *hurts* average performance (-0.59 pp), reducing accuracy in 6 of 9 cells, while the validated rubric improves *all nine* cells ($+1.55$ pp average), with the largest gains on SpreadsheetBench ($+2.3$ to $+3.7$ pp). These results demonstrate the effectiveness of our validated rubric and the resulting meta-skill, which plug directly into any extractor's system prompt without modifying the underlying extraction pipeline.

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 that reliably distinguish higher- from lower-utility skills, and using this rubric as a meta-skill to guide the extractor consistently improves the quality of the generated skills—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 all evaluated cells and plugs directly into any extractor's system prompt. Together, these contributions move agent skill extraction from a heuristic, intuition-driven practice toward a principled, utility-grounded discipline.

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] 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.

[16] 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.

[17] 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.

[18] 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.

[19] Fangzhou Li, Pagkratios Tagkopoulos, and Ilias Tagkopoulos. Skillflow: Scalable and efficient agent skill retrieval system. *arXiv e-prints*, pages arXiv–2504, 2025.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

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

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

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

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

[29] 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. 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 is what enables the controlled cross-extractor and cross-target comparisons that ground all of our findings. We see two natural directions for future work: scaling to richer agent harnesses (for example, with retrieval, planning, or tool-use scaffolds), and scaling to substantially larger skill libraries containing many fine-grained skills, where additional questions of skill selection, composition, and interference become first-class concerns. We view these directions as complementary to the present study rather than as gaps in it, and as promising avenues for building on the utility-grounded foundation established here.

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. For each domain $\mathcal{D}$, we split task instances 1:1 into an experience-generation split $Q_{\mathcal{D}}^{\text{train}}$ and a held-out test split $Q_{\mathcal{D}}^{\text{test}}$. When an official training split is provided by the benchmark, $Q_{\mathcal{D}}^{\text{train}}$ is sampled from it at the same 1:1 proportion relative to $Q_{\mathcal{D}}^{\text{test}}$; otherwise we partition the available instances uniformly at random with a fixed seed. The same splits are used across all (extractor, target) combinations within a domain to ensure that observed differences in Δ are attributable to the extraction side rather than to evaluation noise.

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.

```
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.
```

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

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 `[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.

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

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.

B.5 Extraction Hyperparameters

All extraction experiments use the mode-based method with the default settings listed in Table 7 unless otherwise noted:

In the *map* phase, each trajectory is independently analyzed to extract up to 3 behavioral modes (success or failure patterns). In the *reduce* phase, modes 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 [29] 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.

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

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.

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.

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 5/6 targets ($p < 0.005$) with σ-ratios well above 1.

D 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**.

```
## 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 5 Single-skill injection prompt template.

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.

```
## 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.
```

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

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

Table 7 Default extraction hyperparameters.

E Pairwise Skill Evaluation

The unguided pairwise evaluation (Section 5.2) uses GPT-5.4 and 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.

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 $|\Delta|$ gap exceeds 0.5 pp. This threshold excludes near-ties where the ground-truth ranking is unreliable due to evaluation noise.

F Alternative Harness Evaluation

To verify that our SpreadsheetBench results are not artifacts of the Python-script evaluation harness used in the main experiments, we re-evaluate a subset of conditions using two alternative agentic harnesses: Claude Code (CC) and Codex. These harnesses execute spreadsheet tasks

	Format		Extractor	
Target	p	σ-r	p	σ-r
GPT-5.4	.983	0.11	<.001	**1.98**
GPT-5.2	.436	0.77	<.001	**4.53**
GPT-5.4-mini	.916	0.15	<.001	**1.31**
GPT-5.4-nano	.881	0.40	.003	**1.51**
Qwen3.5-35B	.345	0.67	<.001	**2.54**
Qwen3.5-9B	.452	0.53	.515	0.83

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

```
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"}
```

Table 9 Unguided pairwise judge prompt template.

via interactive tool-use rather than a fixed script, providing an independent check on skill utility under different execution environments. Table 10 reports the resulting Δ matrix.

Target	Base	GPT-5.4	GPT-5.4-mini	Qwen3.5-35B	TE
CC Opus 4.6	70.0	+1.0	+1.0	−3.5	−0.5
CC Sonnet 4.6	66.5	+4.0	+1.0	+2.0	+2.3
Codex GPT-5.4	53.5	+4.0	−5.5	+3.5	+0.7
Codex GPT-5.4-mini	42.5	−1.0	+0.0	−2.0	−1.0
EE		**+2.0**	−0.9	+0.0	

Table 10 SpreadsheetBench Δ (pp) with alternative agentic harnesses (Claude Code / Codex). Green: $\Delta > 0$; Red: $\Delta < 0$.

The overall pattern is consistent with the main results: skill injection yields modest positive gains on average ($\overline{\Delta} = +0.4$ pp), with substantial variance across targets. Notably, stronger targets (CC Opus, Codex GPT-5.4) show positive transfer from GPT-5.4-extracted skills, while the weakest target (Codex GPT-5.4-mini) shows no benefit, echoing the consumption-ability gradient observed in the main experiments.

G Meta-Skill Guidance

Table 11 reports the full per-cell accuracy numbers underlying Figure 5 in Section 6, including the no-skill baseline and the original (un-guided) skill condition.

Domain	Target	No Skill	Original	Guidance: Plausibility (7-dim)	Guidance: Validated (3-dim)	Δ vs Original: Plaus.	Δ vs Original: Valid.
ALFWorld	GPT-5.4	68.66	75.12	74.13	**76.12**	−0.99	+1.00
	Gemini	87.56	88.31	87.81	**88.81**	−0.50	+0.50
	Qwen3.5-35B	57.21	53.73	53.50	**54.11**	−0.23	+0.38
Spreadsheet-Bench	GPT-5.4	37.17	46.17	40.17	**48.50**	−6.00	+2.33
	Gemini	37.50	33.17	34.33	**36.75**	+1.16	+3.58
	Qwen3.5-35B	23.83	29.33	31.49	**33.02**	+2.16	+3.69
SWE-bench	GPT-5.4	68.40	69.72	68.64	**70.10**	−1.08	+0.38
	Gemini	66.53	69.33	70.96	**70.58**	+1.63	+1.25
	Qwen3.5-35B	52.92	55.00	53.51	**55.87**	−1.49	+0.87
Average (9 cells)						−0.59	+1.55

Table 11 **Effect of meta-skill guidance on downstream skill utility** (accuracy %). The plausibility-based rubric (all 7 dimensions, unscreened) hurts on average; the utility-validated rubric (3 screened dimensions) improves all nine cells.

H Contrastive Skill Analysis

H.1 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 12 lists the seven dimensions used as the unguided baseline in Section 6.

#	Dimension
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

Table 12 Seven dimensions of the **plausibility rubric**, generated directly by GPT-5.4 without any pair-level or utility-level grounding.

H.2 Raw Rubric (from Contrastive Pipeline)

The contrastive analysis (Section 6) synthesized seven candidate quality dimensions from recurring themes across 17 high-gap skill pairs. Table 13 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).

H.3 Representative Contrastive Cases

Tables 14 and 15 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.

#	Dimension	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%

Table 13 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.

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

Higher-Δ skill (Δ= + 7.5, ext = Gemini-3.1-Pro)	**Lower-Δ skill** (Δ= + 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 → Acquire → Transform → Navigate → 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	
✓	✗

Table 15 Contrastive case: ALFWorld, target = GPT-5.4, Δ gap = 6.0 pp.