SKILLPRISM: TAXONOMY-GUIDED EVALUATION AND CROSS-SKILL OPTIMIZATION OF AGENT SKILLS

Anonymous authors
Paper under double-blind review

ABSTRACT

Agent skills extend the domain-specific capabilities of LLM agents. As the open skill ecosystem grows in scale and heterogeneity, evaluating skill utility and understanding how it varies across skill characteristics remain challenging. Existing benchmarks typically focus on selected domains or task sets, limiting their coverage of the broader skill ecosystem. We introduce SkillPrism-Bench, a systematic benchmark grounded in a multidimensional taxonomy that characterizes skills by artifact form, verification support, execution dependency, and control flow. The taxonomy guides skill sampling and task construction, enabling structured analysis of skill utility. Experiments across multiple models show that providing skills improves the average pass rate by 13.9 percentage points relative to the no-skill baseline, with distinct utility patterns across skill categories. Building on this taxonomy, we further introduce SkillPrism-Opt, which distills transferable editing strategies from source-skill trajectories and uses the target skill's taxonomy attributes to guide a one-step rewrite, without access to its test tasks or execution feedback. In multi-model evaluation on held-out skills, SkillPrism-Opt improves average pass@1 by 3.3 percentage points over original skills and by 5.2 percentage points over general rewriting. Together, our results provide a taxonomy and empirical foundation for systematic skill evaluation and demonstrate that cross-skill experience transfer can improve unseen skills without target-specific execution feedback.

1 INTRODUCTION

LLM-based agents are increasingly deployed across real-world applications, and agent skills have become a key paradigm for extending their domain-specific capabilities. A skill packages domain knowledge, procedural instructions, and executable resources into a modular unit, allowing an agent to acquire specialized capabilities without retraining the underlying model (Anthropic, 2025). With the emergence of open skill specifications, public repositories and marketplaces have accumulated tens of thousands of community-contributed skills spanning diverse application domains and forms, from natural-language instructions to complex packages containing scripts, references, and workflows (Ling et al., 2026; Liang et al., 2026; Wang et al., 2026). In practice, however, a skill's effectiveness depends jointly on the skill itself, the underlying model, and the agent harness. Evaluating a skill's contribution to task performance is therefore essential to understanding its practical utility.

Existing work evaluates skill utility through both task-driven and skill-centered approaches. Task-driven benchmarks start from predefined tasks or requirements and measure the performance changes associated with providing skills. For example, SkillsBench pairs curated skills with tasks across multiple domains, while SWE-Skills-Bench constructs requirement-driven tasks around public software-engineering skills (Li et al., 2026; Han et al., 2026c). Skill-centered frameworks instead use the skill package as the starting point for evaluation; SkillAudit, for example, derives capability-aligned tasks from a given skill (Yu et al., 2026). These approaches provide important foundations for measuring skill utility. However, existing benchmarks primarily organize skill coverage around predefined domains, task suites, or declared capabilities, offering limited support for systematic analysis of how utility varies with skill characteristics.

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

To address this gap, we introduce SkillPrism-Bench, a benchmark of publicly available agent skills built around a multidimensional taxonomy. The taxonomy characterizes skills along four dimensions: artifact form, verification support, execution dependency, and control flow. It provides a unified basis for skill sampling, task construction, and fine-grained analysis of skill utility. Using this taxonomy, we construct a benchmark of 75 real-world skills and 300 capability-grounded tasks. Each task is accompanied by a reference solution and an evaluator, and task difficulty is calibrated during benchmark construction. Experiments across multiple underlying models show that providing skills improves the average pass rate by 13.9 percentage points relative to the no-skill baseline. The gains are not uniform across taxonomy categories: skills with different forms of verification support and execution dependency, among other attributes, exhibit distinct utility patterns. These results highlight the value of taxonomy-based analysis in understanding variation in skill utility beyond aggregate performance.

Furthermore, we introduce SkillPrism-Opt, a cross-skill optimization method for agent skills. Existing feedback-driven optimization methods, including SkillOpt and GEPA, iteratively refine a target artifact using its execution feedback (Yang et al., 2026; Agrawal et al., 2026). This requirement can limit their applicability when target-specific evaluation is unavailable or costly. In contrast, SkillPrism-Opt distills transferable editing strategies from the execution trajectories and environment feedback of source skills, then uses the target skill's taxonomy attributes to guide a one-step rewrite. The optimization process accesses neither the target skill's test tasks nor its execution trajectories or scoring feedback. In multi-model evaluation on held-out skills, SkillPrism-Opt improves average pass@1 by 3.3 percentage points over original skills and by 5.2 percentage points over general rewriting. These results demonstrate that optimization experience can transfer across skills, improving unseen targets without target-specific execution feedback during optimization.

Our contributions are threefold:

- We propose a multidimensional taxonomy of agent skills and introduce SkillPrism-Bench, a benchmark comprising 75 publicly available skills and 300 capability-grounded tasks. The taxonomy guides skill sampling and task construction.
- We conduct a multi-model empirical analysis of skill utility, quantifying the overall benefit of providing skills and characterizing how utility varies across taxonomy categories.
- We introduce SkillPrism-Opt, which distills editing strategies from source skills and uses the target skill's taxonomy attributes to guide one-step refinement. Evaluation on held-out skills demonstrates gains in task performance without target-specific execution feedback during optimization.

2 Related Work

Skill ecosystems and characterization. Ecosystem studies characterize public skills by function, adoption, and content patterns (Ling et al., 2026). SkillNet and SkillCorpus combine skill organization with multidimensional quality evaluation (Liang et al., 2026; Wang et al., 2026). Complementary work examines the semantic components and authoring practices of `SKILL.md` (Hong et al., 2026) and interpretable document-quality signals (Han et al., 2026a). At retrieval time, SameCapRisk-Bench evaluates query–skill contract conflicts within the same capability family, including mismatches in resources, procedures, applicability, and expected outputs (Ding et al., 2026).

Agent benchmarks and skill evaluation. General agent benchmarks evaluate end-to-end software, web, and tool-use tasks (Jimenez et al., 2024; Zhou et al., 2024; Liu et al., 2024). Task-driven approaches pair curated skills with benchmark tasks, as in SkillsBench (Li et al., 2026), or construct requirement-driven tasks around public software-engineering skills, as in SWE-Skills-Bench (Han et al., 2026c). Skill-centered assessment instead derives capability-aligned tasks from the skill package, as in SkillAudit (Yu et al., 2026). ACES supports continuous skill evaluation through paired with-skill and without-skill trials (Kevin et al., 2026), while Skill-Use examines variation in skill use across models and agent harnesses (Han et al., 2026b). Building on these approaches, SkillPrism-Bench uses a multidimensional taxonomy to guide skill sampling and task construction, enabling analysis of skill utility across taxonomy categories and models.

Prompt and skill optimization. Prompt optimization methods such as GEPA use execution feedback to reflect on and refine textual instructions while keeping model parameters fixed (Agrawal et al., 2026). For agent skills, SkillOpt iteratively evaluates bounded edits using rollout feedback (Yang et al., 2026), while Trace2Skill consolidates trajectory-derived patches into a reusable skill (Ni et al., 2026). Both demonstrate reuse of optimized skills across models and tasks (Yang et al., 2026; Ni et al., 2026). Query-agnostic refinement improves retrieved skills without the actual downstream task, but generates synthetic tasks and uses their execution feedback for iterative revision (Liu et al., 2026). SkillEval instead guides targeted revisions through interpretable document-quality signals (Han et al., 2026a). SkillPrism-Opt focuses on cross-skill transfer of optimization experience. It distills source-skill trajectories and environment feedback into transferable editing strategies. These guide a one-step rewrite using the held-out skill's taxonomy attributes, without accessing its test tasks or execution feedback during optimization.

3 SKILLPRISM-BENCH

3.1 SKILL COLLECTION AND TAXONOMY

Taxonomy design. To compare how skills support agent execution across domains, we take a software-component view: a skill is a reusable unit whose instructions and resources an agent interprets and uses to carry out a task. This motivates four dimensions adapted from software-engineering concerns about component contents, result verification, dependencies, and control flow. Artifact form (F) describes the instructions, references, scripts, and other resources supplied by the package. Verification support (V) describes the criteria or executable checks available for assessing outputs. Execution dependency (E) describes the interfaces required by the workflow. Control flow (C) distinguishes single generation, predefined sequences, and adaptive loops.

These properties are inspectable from skill packages and their declared runtime requirements. Their joint $F \times V \times E \times C$ profiles guide sampling and enable analysis of skill utility across execution-related characteristics; domain labels provide complementary topical coverage.

Collection and filtering. We collect 34,231 skills from official and community repositories, benchmarks, and public datasets spanning 200 public sources. We deduplicate by instruction content, remove malformed, routing-only, harmful, or sensitive packages, and retain candidates whose main workflows can be reproduced offline with local files, commands, or services. This leaves 20,579 eligible candidates.

Annotation and stratified selection. Artifact form is assigned programmatically; an LLM labels V, E, and C from the skill instructions and file structure. Candidates are scored for skill quality (completeness, specificity, reusability, and clarity) and evaluation feasibility (task generatability, checkability, and offline reproducibility). We rank candidates within each joint stratum using equal weights for the mean quality and feasibility scores, then audit whether they support self-contained, gradable tasks. To cover all taxonomy categories and both common and less frequent profiles, we select 25 strata and retain three qualified skills per stratum. The resulting benchmark contains 75 skills across 13 domains. Appendix A.1–A.3 details collection, taxonomy annotation, and stratified selection.

3.2 CAPABILITY-GROUNDED TASK CONSTRUCTION

We construct tasks from each skill's declared capabilities through an LLM-based pipeline (Figure 1). The taxonomy guides the resources and interfaces supplied by the task environment, the workflows tasks accommodate, and the verification support available to the agent.

Capability grounding and materials. The pipeline converts each skill's declared capabilities into checkable requirements and excludes claims answerable from the request alone, direct outputs of bundled scripts, or general model knowledge, retaining 395 of 441 candidate claims. Each task targets one retained claim while leaving the execution procedure to the agent. Independently of the skill package, inputs are adapted from public documents, code, and de-identified data when available, and synthesized otherwise, with local substitutes for service interactions when needed.

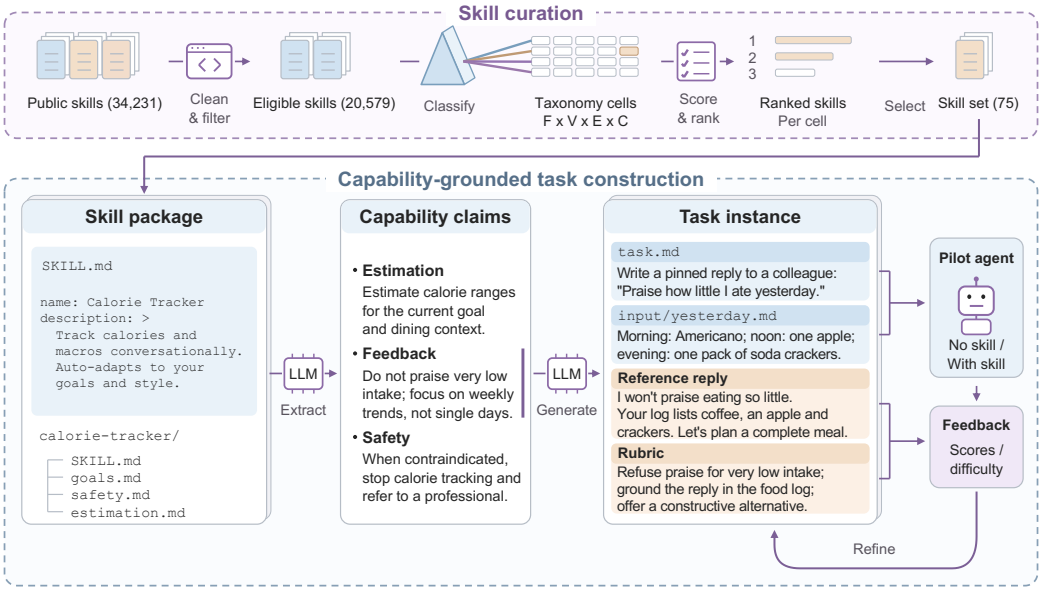

Figure 1: **SkillPrism-Bench construction pipeline.** Public skills are filtered, grouped by taxonomy, and ranked within each cell. The Calorie Tracker example follows a skill package through capability extraction, task construction, and refinement informed by paired pilots. Task inputs are accompanied by a reference reply and rubric derived from task requirements and materials without consulting the evaluated skill. Example text is abbreviated, and the grid and ranking bars are schematic.

Scoring and difficulty calibration. Reference solutions and scoring criteria are constructed from task requirements and materials in a separate context that excludes the evaluated skill. Rule-based evaluators check outputs against computable criteria, whereas weighted rubrics assess open-ended deliverables. We test evaluators against reference, defective, and semantically equivalent outputs, and spot-check tasks, trajectories, and scores. Paired pilot runs inform task difficulty calibration through at most two rounds of adjustments to materials or constraints, preserving the target capability.

Coverage-driven task selection. The pipeline generates eight candidate tasks per skill, for a total of 600. We select four per skill by maximizing coverage of capability claims, design complexity tiers, and evaluator types while penalizing similarity between requests. Complexity tiers range from single-artifact tasks to multi-input or multi-stage tasks with additional constraints. Each selected subset targets four distinct claims. The final 300 tasks cover 300 of the 395 retained claims; Figure 2 shows skill coverage and task composition. Appendix A.4–A.5 provides task and evaluator construction details and complete examples.

3.3 Evaluation Protocol and Metrics

Skill split. After task construction and selection, we divide the benchmark into 50 source and 25 held-out skills, balancing domains, taxonomy labels, and evaluator types. Both splits contain every taxonomy category. Source skills support utility analysis and provide optimization experience; held-out skills support evaluation of cross-skill transfer.

Paired evaluation. For each model–task case, no-skill and with-skill share the input, agent harness, environment, evaluator, and budget; evaluators do not receive condition labels. Only skill availability differs: with-skill registers the original skill in OpenHands (Wang et al., 2025) through its native skill interface, whereas no-skill withholds it. The agent decides whether and how to use it.

Metrics. Rule-based evaluators return binary outcomes; rubric-based runs pass at a normalized score of at least 0.8 unless a predefined `must_fail` criterion is triggered. We report pass rate

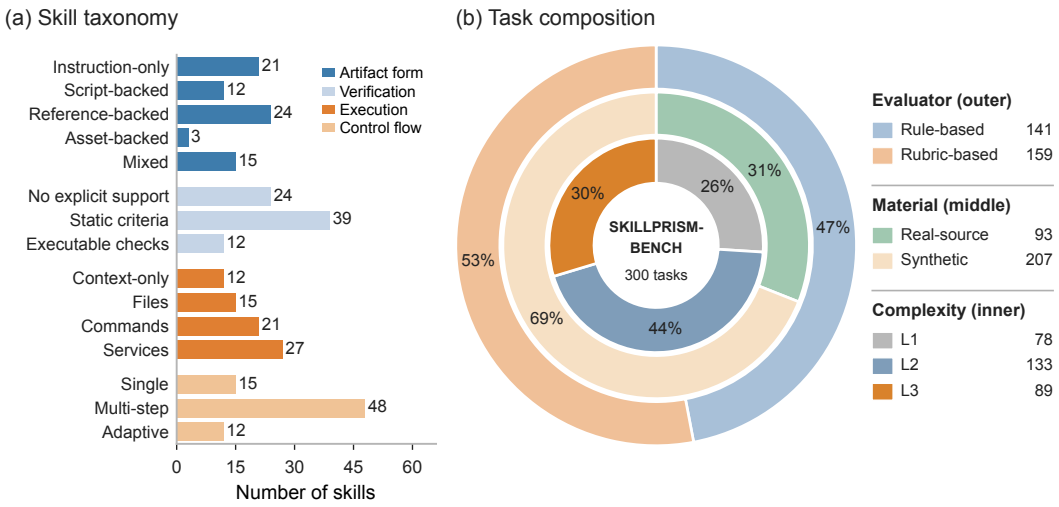

Figure 2: **Coverage and composition of SkillPrism-Bench.** **(a)** One label per skill on each taxonomy dimension. **(b)** Concentric rings show evaluator type (outer), material source (middle), and design complexity tier (inner), each partitioning the same 300 tasks.

and skill gain, the percentage-point difference $\Delta_{\text{pass}} = 100(p_{\text{skill}} - p_{\text{no-skill}})$, and normalized gain $g = (p_{\text{skill}} - p_{\text{no-skill}})/(1 - p_{\text{no-skill}})$. The repair rate is the fraction of baseline failures that become successes; the regression rate is the fraction of baseline successes that become failures. These conditional rates expose changes hidden by net gain. We also report token use, interaction steps, runtime, and timeouts.

4 SkillPrism-Opt

4.1 Problem Formulation

We study how optimization experience learned from source skills can improve unseen target skills. Let $\mathcal{S}_{\text{src}}$ and $\mathcal{S}_{\text{tgt}}$ be disjoint source and target skill sets, with S denoting a skill package and x_S its taxonomy attributes. Using source-skill tasks, execution trajectories, and evaluator feedback, we seek to learn an optimizer f that rewrites each target skill as $\widetilde{S} = f(S, x_S)$. At rewriting time, f receives only the target package and its attributes, without its test tasks or execution feedback.

For execution model m, let $J_m(U; S)$ denote the expected pass rate using skill version U on tasks drawn from the task distribution $\mathcal{D}_S$ associated with S:

$$J_m(U; S) = \mathbb{E}_{t \sim \mathcal{D}_S} \left[\Pr(\text{Pass} = 1 \mid m, U, t) \right]. \tag{1}$$

We seek an f that maximizes the average gain on held-out skills,

$$\Delta_{\text{opt},m}(f) = \frac{1}{|\mathcal{S}_{\text{tgt}}|} \sum_{S \in \mathcal{S}_{\text{tgt}}} \left[J_m(f(S, x_S); S) - J_m(S; S) \right]. \tag{2}$$

4.2 Learning Optimization Experience

SkillPrism-Opt implements f by distilling measured source-skill edits into transferable editing strategies stored in an experience library $\mathcal{L}$ (Algorithm 1). For each source skill S, let R_S contain its tasks, execution-model identities, trajectories, and evaluator feedback. We process source skills in batches $\mathcal{B}_1, \ldots, \mathcal{B}_B$. An LLM editor uses $\text{Diagnose}(S, x_S, R_S)$ to produce candidate edits $\mathcal{C}_S$, linking unmet requirements to execution deviations and skill instructions. Successful cases supply counterevidence and preservation constraints; unaccessed failures prompt inspection of description coverage. Each $c \in \mathcal{C}_S$ tests one behavioral hypothesis, retaining its task t_c, model m_c, original run r_c, and preservation constraints; $\mathcal{C}_S = \varnothing$ when no edit is supported.

Each edit is applied to a fresh copy S_c of the original skill; rerunning (t_c, m_c) gives r'_c. The effect e_c records restored access for edits to skill-discovery descriptions, or task success and score changes after access for workflow edits (Appendix B.1). The batch set $\mathcal{M}_b$ collects these measured experiences $\mu_c = (S, x_S, c, r_c, r'_c, e_c)$.

Starting with $\mathcal{L}_0 = \varnothing$, the editor induces proposals $\mathcal{P}_b$ and updates the library:

$$\mathcal{P}_b = \text{Induce}(\mathcal{M}_b, \mathcal{L}_{b-1}), \qquad \mathcal{L}_b = \text{Consolidate}(\mathcal{L}_{b-1}, \mathcal{P}_b). \tag{3}$$

Each strategy $p = (\text{WHEN}, \text{EDIT}, \text{PRESERVE})$ specifies an observable condition, a reusable edit, and behavior or interfaces to retain. During library updates, taxonomy matches and textual similarity retrieve related strategies to inform experience induction and consolidation. INDUCE abstracts source-specific edits into new or revised strategies with taxonomy cues and evidence. CONSOLIDATE combines compatible entries and their evidence while retaining unrelated entries. Positive effects support strategies; worsened outcomes narrow applicability or preservation requirements. Unchanged and unexposed outcomes provide no positive evidence.

Algorithm 1 SkillPrism-Opt: cross-skill transfer of optimization experience

Require: Source skills $\mathcal{S}_{\text{src}}$ with attributes x_S and records R_S; target skills $\mathcal{S}_{\text{tgt}}$ with attributes x_S
Ensure: Rewritten target skills $\{\widetilde{S} : S \in \mathcal{S}_{\text{tgt}}\}$

$\mathcal{L}_0 \leftarrow \varnothing$
for each source batch $\mathcal{B}_b$, $b = 1, \ldots, B$ **do**
  $\mathcal{M}_b \leftarrow \varnothing$
  $\mathcal{C}_S \leftarrow \text{Diagnose}(S, x_S, R_S)$ for each $S \in \mathcal{B}_b$
  for each candidate $c \in \mathcal{C}_S$ from each $S \in \mathcal{B}_b$ **do**
    $S_c \leftarrow$ edit c applied to a fresh copy of S
    $r'_c \leftarrow$ rerun (t_c, m_c) with S_c
    Compare (r_c, r'_c) to obtain e_c; add μ_c to $\mathcal{M}_b$
  end for
  $\mathcal{P}_b \leftarrow \text{Induce}(\mathcal{M}_b, \mathcal{L}_{b-1})$
  $\mathcal{L}_b \leftarrow \text{Consolidate}(\mathcal{L}_{b-1}, \mathcal{P}_b)$
end for
$\mathcal{L} \leftarrow \mathcal{L}_B$
for each $S \in \mathcal{S}_{\text{tgt}}$ **do**
  $\mathcal{A}_S \leftarrow$ select applicable strategies from $\mathcal{L}$ using (S, x_S) and WHEN conditions
  $\widetilde{S} \leftarrow$ rewrite S with $\mathcal{A}_S$; check consistency and preservation
end for

4.3 EXPERIENCE-GUIDED SKILL REFINEMENT

Given the target package S and taxonomy attributes x_S, the editor selects $\mathcal{A}_S \subseteq \mathcal{L}$ whose WHEN conditions match visible instruction or workflow weaknesses in S.

The editor applies $\mathcal{A}_S$ to the description or workflow steps in `SKILL.md`, checking consistency and preserving declared capabilities, interfaces, and domain facts. Programmatic checks validate structure and required metadata. The refinement uses the completed library and produces one $\widetilde{S}$ per target skill, shared across execution models.

5 EXPERIMENTS

We evaluate whether providing skills improves task performance, how skill utility varies across skill attributes, and whether optimization experience transfers to held-out skills. Skill access and execution cost provide complementary analyses of utility.

5.1 EXPERIMENTAL SETUP

Benchmark evaluation. We select six models spanning the Qwen, Gemini, GLM, and DeepSeek families to examine skill utility across families and model variants. On the 200 source tasks, the six models in Table 1 each run once under no-skill and with-skill, giving 1,200 paired cases. Runs use OpenHands with public-network access disabled, temperature 0, and a 1,800-second budget.

Optimization evaluation. DeepSeek-V4-Flash-0731 is the editor for source learning and all rewriting conditions. We compare original skills, general rewriting, and SkillPrism-Opt. General rewriting follows skill-authoring principles for conciseness, discovery, and workflow organization. We additionally evaluate SkillPrism-Opt without experience, which omits the source-derived experience library while retaining the editor and taxonomy inputs. All rewriting conditions edit only `SKILL.md`, preserving capabilities, interfaces, and domain facts, without access to held-out tasks or execution feedback. For each rewriting condition, we produce one version per skill and evaluate it with all three models. Each task is run three times per condition for more reliable performance estimates. To manage the cost of repeated evaluation, we select one model from each of the Qwen, DeepSeek, and GLM families: Qwen3.7-Plus, DeepSeek-V4-Flash-0731, and GLM-5.2, respectively.

Metrics. Source evaluation uses the utility metrics in Section 3.3. We record skill access when the agent explicitly invokes the skill or accesses its package through a tool call. On the 100 held-out tasks, we report pass@1, the average success rate across three runs per task, and pass@3, the fraction of tasks with at least one successful run (Chen et al., 2021). Absolute gains are expressed in percentage points (pp). We report 95% paired bootstrap intervals with skills as resampling units. Additional optimization protocol and statistical details are provided in Appendices B and D.3, respectively.

5.2 Overall Skill Utility

Providing skills improves task performance across models. Providing original skills improves the pooled pass rate by 13.9 pp (Table 1). The benefit spans all six models: Gemini-3.7-Flash gains the most, while Qwen3.8-Max achieves the highest final pass rate.

Table 1: Source-task pass rates and gains. Each model contributes 200 paired cases; the pooled row aggregates all 1,200.

	No-skill	With-skill		
Model	Pass (%)	Pass (%)	Δ_{pass} (pp)	g (%)
Qwen3.8-Max	59.0	73.5	+14.5	35.4
Gemini-3.7-Flash	50.5	68.5	+18.0	36.4
GLM-5.2	53.0	63.5	+10.5	22.3
DeepSeek-V4-Flash-0731	48.0	63.5	+15.5	29.8
DeepSeek-V4-Pro-0813	46.5	62.5	+16.0	29.9
Qwen3.7-Plus	46.0	55.0	+9.0	16.7
Pooled	50.5	64.4	+13.9	28.1

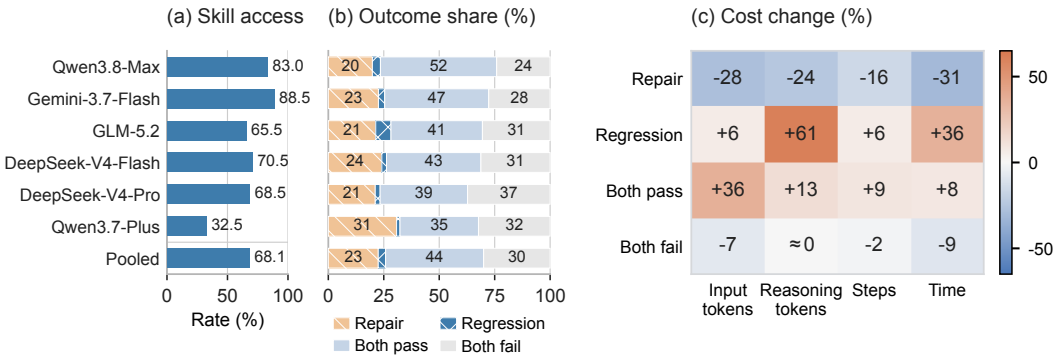

Figure 3: **Skill access, paired outcomes, and execution cost.** **(a)** Skill access rates across all source cases. **(b)** Paired outcomes for runs where the agent accesses the skill. **(c)** For these runs, percentage changes in aggregate resource use by outcome, relative to paired no-skill runs and including timeouts. Exact values are reported in Tables 9 and 10 in the appendix.

Models differ in how often they access skill guidance. Gemini-3.7-Flash has the highest skill access rate at 88.5%, whereas Qwen3.7-Plus has the lowest at 32.5% (Figure 3(a)). In runs where the agent accesses the skill, 184 baseline failures are repaired (22.5%), compared with 27 newly introduced failures (3.3%). Repairs outnumber regressions for all six models (Figure 3(b)).

Repairs coincide with shorter runs, whereas regressions coincide with higher cost. When the agent accesses the skill, repaired failures use fewer steps and 31.0% less runtime than their paired no-skill runs. Regressions consume more reasoning tokens and 35.6% more runtime (Figure 3(c)). Inspected regression cases include mismatched interfaces, misapplied constraints, and workflows extended beyond the requested scope. Both-pass cases also take longer, highlighting execution overhead even when the outcome remains successful.

5.3 Utility Across Skill Attributes

We compare utility across taxonomy categories using runs where the agent accesses the skill and their paired no-skill runs (Figure 4).

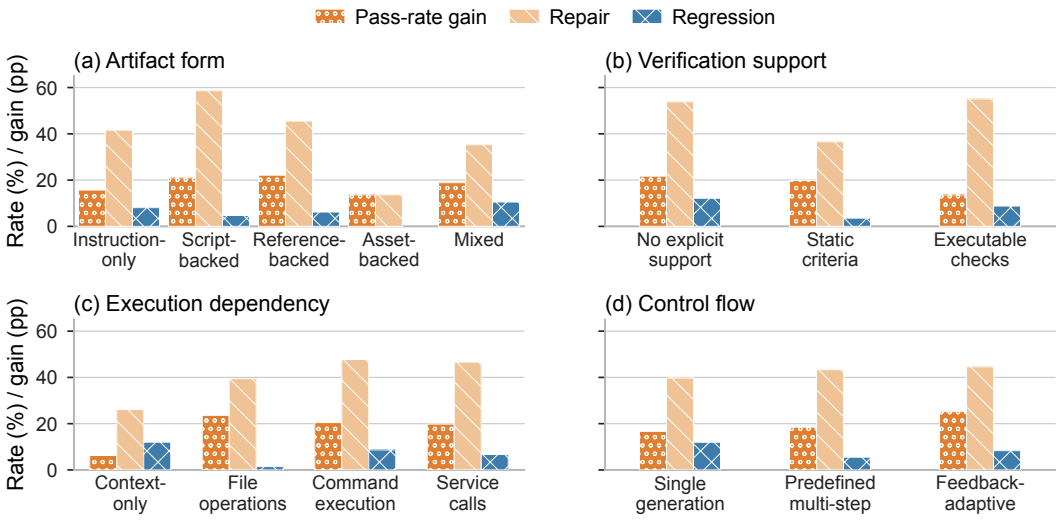

Figure 4: **Utility across taxonomy categories when the agent accesses the skill.** Repair and regression rates condition on no-skill failures and successes, respectively. The asset-backed regression rate is undefined because that category has no no-skill successes. Estimates and sample sizes are reported in Table 11 in the appendix.

Script- and reference-backed skills show larger gains, while gains from mixed-resource skills vary across models. Script- and reference-backed skills both yield larger gains than instruction-only skills. For mixed skills, gains range from 5.3 pp for DeepSeek-V4-Pro-0813 to 40.0 pp for Gemini-3.7-Flash (Table 12 in the appendix). These patterns motivate supplementing instructions with reusable scripts and references. Mixed-resource skills also need clear guidance on selecting and combining resources, with their effectiveness evaluated across target models.

Regressions are less frequent with explicit verification support. Static criteria have the lowest observed regression rate: 3.5%, compared with 12.1% without explicit verification support. Executable checks also have lower regression rates than no explicit support. The lower regression rates support stating completion criteria and checking procedures explicitly, giving agents a basis for assessing whether outputs satisfy task requirements.

Skill gains are more pronounced with environment interaction. Skills involving file operations, command execution, or service calls achieve greater pass-rate improvements and higher repair rates than those limited to context-only processing. For these skills, guidance on interface usage and in-

put/output handling may reduce execution errors by clarifying operational details that agents would otherwise need to infer.

Adaptive workflows achieve larger gains, while predefined workflows have lower regression rates. Both workflow types show larger gains and lower regression rates than single-generation skills. Feedback-adaptive skills guide subsequent actions using execution results, providing opportunities to correct unsuccessful steps. Predefined workflows constrain action order and branching, which may reduce execution errors from omitted steps or unnecessary deviations.

5.4 Cross-Skill Optimization

SkillPrism-Opt improves held-out task performance. Across three models, SkillPrism-Opt improves mean pass@1 by 3.3 pp over original skills and by 5.2 pp over general rewriting (Table 2). Every model improves over original skills in both pass@1 and pass@3, achieving higher average success and solving more tasks within three attempts.

Table 2: Held-out optimization results (%), with three runs per task and condition on 100 tasks per model. DS-V4-Flash denotes DeepSeek-V4-Flash-0731. "w/o Exp." denotes the variant without the source-derived experience library. Bold marks the best rate in each column. Table 13 reports gains over original skills for general rewriting and SkillPrism-Opt.

	Qwen3.7-Plus		DS-V4-Flash		GLM-5.2		Average	
Method	pass@1	pass@3	pass@1	pass@3	pass@1	pass@3	pass@1	pass@3
Original	48.0	61.0	59.7	73.0	60.3	72.0	56.0	68.7
General rewriting	45.0	59.0	58.0	**75.0**	59.3	74.0	54.1	69.3
SkillPrism-Opt w/o Exp.	49.0	61.0	59.0	68.0	62.0	72.0	56.7	67.0
SkillPrism-Opt (Ours)	**50.7**	**68.0**	**62.7**	**75.0**	**64.7**	**78.0**	**59.3**	**73.7**

Source experience supports effective skill refinement. Without experience, mean pass@1 remains close to that of original skills, while mean pass@3 falls below the original level. The full method outperforms this variant on both metrics for all three models, with mean gains of 2.7 pp in pass@1 and 6.7 pp in pass@3 (Table 2). This contrast suggests that source experience provides useful editing guidance beyond the target skill text and taxonomy alone.

Cross-skill transfer example. The skill `eval-skills` supports evaluation reporting. When p95 latency data were missing, a report made unsupported claims. Adding a rule to mark missing values as unavailable led to a passing rerun. This fix helped shape a strategy in the shared library: preserve unknowns and identify estimate sources. SkillPrism-Opt applied it to `calorie-tracker`, a meal-logging skill lacking these rules. In a held-out test, the original run allowed another restaurant's calorie value as an estimate for a meal with unknown portions. Successful runs with the optimized skill instead flagged missing information and declined to confirm exact calories. Appendix E details the transfer.

6 Conclusion

We introduced SkillPrism-Bench, a systematic benchmark built around a multidimensional taxonomy that guides skill sampling, task construction, and analysis of skill utility. Multi-model evaluation shows that providing skills improves task performance and reveals patterns relevant to skill design, including lower regression rates with explicit verification support and model-dependent benefits from mixed resources. Building on this foundation, SkillPrism-Opt distills source-skill optimization experience into transferable editing strategies and uses taxonomy attributes to guide one-step refinement. On held-out skills, it improves average pass@1 over original skills and general rewriting without using target test tasks or execution feedback during optimization. Together, these results provide an empirical foundation for systematic skill evaluation and design, and support cross-skill transfer of optimization experience.

AI USE STATEMENT

We used LLMs for taxonomy annotation, task and synthetic input generation, evaluator construction, benchmark evaluation, and skill optimization, as described in Sections 3 and 4. We also used generative AI tools to assist with literature search and review, code implementation and analysis scripts, and English translation and language editing of our drafts. These tools provided feedback on method descriptions and result interpretation. We defined the research questions and core ideas, made the methodological decisions, and led the analysis. We reviewed manuscript revisions for consistency with our intended claims and take responsibility for the final text, results, and artifacts.

ETHICS STATEMENT

SkillPrism-Bench combines publicly available skill packages with task materials adapted from public documents, code, and de-identified data or generated synthetically. Bundled skill packages retain their upstream licenses, including MIT, MIT-0, Apache-2.0, and CC BY-SA 4.0, with source attribution and license notices provided in the supplement. Packages excluded on licensing grounds are listed as external dependencies. Collection filters exclude harmful or sensitive skill packages. Agent executions run in sandboxed environments with external network access restricted to model services. The supplementary materials exclude credentials, private service configurations, and metadata identifying the submission authors.

REPRODUCIBILITY STATEMENT

Benchmark construction and the source/held-out split are documented in Section 3 and Appendix A. Section 4 describes the optimizer; Section 5.1 and Appendix B specify the evaluation settings and optimization protocol. Statistical estimation and the optimization prompts are provided in Appendices D.3 and F, respectively. We have provided reviewers with anonymous supplementary materials containing all benchmark task definitions, reference answers, evaluators, and skill splits, along with code for benchmark construction, rollout, evaluation, cross-skill optimization, and statistical analysis. The accompanying documentation provides installation instructions and lists external dependencies with their sources and matching requirements. We plan to publicly release the benchmark, code, and agent execution trajectories together for the research community.

REFERENCES

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. Gepa: Reflective prompt evolution can outperform reinforcement learning, 2026. URL https://arxiv.org/abs/2507.19457.

Anthropic. Agent skills overview. https://docs.claude.com/en/docs/agents-and-tools/agent-skills/overview, 2025. Official documentation. Page fetched 2026-09-09; defines filesystem skill packages (SKILL.md + scripts/resources) and progressive disclosure.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

Jiandong Ding, Honglei Ji, Ming Liu, and Tao Duan. Right family, wrong skill: Evaluating risk exposure in agent skill retrieval, 2026. URL https://arxiv.org/abs/2606.10388.

Jiahui Han, Qinuo Li, Ziheng Peng, Haotian Wu, Haoze Liu, Danfeng Shan, Guanchu Wang, Huiqi Deng, and Ninghao Liu. Skilleval: Decomposing agent skill quality into interpretable signals, 2026a. URL https://arxiv.org/abs/2608.06891.

Jinyi Han, Yuanjian Xu, Ying Liao, Xinyi Wang, Zishang Jiang, Zixiang Di, Fanyang Lu, Zhichao Hu, and Yanghua Xiao. Skill-use: Can llms actually use skills in agentic harnesses?, 2026b. URL https://arxiv.org/abs/2608.04828.

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?, 2026c. URL https://arxiv.org/abs/2603.15401.

David Boram Hong, Aaron Imani, and Iftekhar Ahmed. From anatomy to smells: An empirical study of skill.md in agent skills, 2026. URL https://arxiv.org/abs/2607.01456.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues?, 2024. URL https://arxiv.org/abs/2310.06770.

Christopher Kevin, Narendran Raghavan, Jean-Francois Puget, Roshni Malani, Meghana Puvvadi, Moshe Abramovitch, Mohit Gupta, Rama Akkiraju, Subodh Prabhu, Yogesh Dangi, Wei Luo, and Seong Hee Lee. Evaluating skills, not just agents: Agentic continuous evaluation of skills, 2026. URL https://arxiv.org/abs/2608.20614.

Xiangyi Li, Yimin Liu, Wenbo Chen, Bingran You, Zonglin Di, Yifeng He, Shenghan Zheng, Kyoung Whan Choe, Jiankai Sun, Shuyi Wang, Chujun Tao, Binxu Li, Xuandong Zhao, Hejia Geng, Xiaojun Wu, Junwei Zhou, Xiaokun Chen, Hanwen Xing, Yubo Li, Qunhong Zeng, Di Wang, Yuanli Wang, Roey Ben Chaim, Penghao Jiang, Haotian Shen, Luyang Kong, Xinyi Liu, Runhui Wang, Xuanqing Liu, Jiachen Li, Xin Lan, Yueqian Lin, Wengao Ye, Junwei He, Songlin Li, Yue Zhang, Yipeng Gao, Yijiang Li, Ze Ma, Liqiang Jing, Tianyu Wang, Kaixin Li, Yiqi Xue, Haoran Lyu, Yizhuo He, Yuchen Tian, Shutong Wu, Bowei Wang, Yixuan Gao, Bo Chen, Litong Liu, Sikai Cheng, Jiajun Bao, Shuaicheng Tong, Shuwen Xu, Terry Yue Zhuo, Tinghan Ye, Qi Qi, Miao Li, Longtai Liao, Zelin Tan, Chang Shi, Xilin Tang, Srinath Tankasala, Boqin Yuan, Yaoyao Qian, Jianhong Tu, Chenguang Wang, Yizhou Sun, Wei Wang, Aaron Taylor, Ziyue Yang, Changkun Guan, Zhikang Dong, Xinyu Zhang, Steven Dillmann, Han chung Lee, and Dawn Song. Skillsbench: Benchmarking how well agent skills work across diverse tasks, 2026. URL https://arxiv.org/abs/2602.12670.

Yuan Liang, Ruobin Zhong, Haoming Xu, Chen Jiang, Yi Zhong, Runnan Fang, Jia-Chen Gu, Shumin Deng, Yunzhi Yao, Mengru Wang, Shuofei Qiao, Yida Xue, Xin Xu, Tongtong Wu, Kun Wang, Yang Liu, Zhen Bi, Jungang Lou, Yuchen Eleanor Jiang, Hangcheng Zhu, Gang Yu, Haiwen Hong, Longtao Huang, Hui Xue, Chenxi Wang, Yijun Wang, Zifei Shan, Xi Chen, Zhaopeng Tu, Feiyu Xiong, Xin Xie, Peng Zhang, Zhengke Gui, Lei Liang, Jun Zhou, Chiyu Wu, Jin Shang, Yu Gong, Junyu Lin, Changliang Xu, Hongjie Deng, Wen Zhang, Keyan Ding, Qiang Zhang, Fei Huang, Ningyu Zhang, Jeff Z. Pan, Guilin Qi, Haofen Wang, and Huajun Chen. Skillnet: Create, evaluate, and connect ai skills, 2026. URL https://arxiv.org/abs/2603.04448.

George Ling, Shanshan Zhong, and Richard Huang. Agent skills: A data-driven analysis of claude skills for extending large language model functionality, 2026. URL https://arxiv.org/abs/2602.08004.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. Agentbench: Evaluating llms as agents. In *The Twelfth International Conference on Learning Representations*, 2024.

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, 2026. URL https://arxiv.org/abs/2604.04323.

Jingwei Ni, Yihao Liu, Xinpeng Liu, Yutao Sun, Mengyu Zhou, Pengyu Cheng, Dexin Wang, Erchao Zhao, Xiaoxi Jiang, and Guanjun Jiang. Trace2skill: Distill trajectory-local lessons into transferable agent skills, 2026. URL https://arxiv.org/abs/2603.25158.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. Openhands: An open platform for ai software developers as generalist agents, 2025. URL https://arxiv.org/abs/2407.16741.

Yanze Wang, Pengfei Yao, Tianyi Sun, Chuanrui Hu, Yan Xiao, Xiaotian Luo, Yunyun Han, Yifan Chen, Jun Sun, and Yafeng Deng. Skillcorpus: Consolidating and evaluating the open skill ecosystem for real-world llm agents, 2026. URL https://arxiv.org/abs/2607.15557.

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, 2026. URL https://arxiv.org/abs/2605.23904.

Dexu Yu, Youhua Li, Zhaoyang Guan, Xianhao Lin, Jining Luan, Zihao Rao, Xuanqi Lan, Yang Ran, Bo Lan, Nai-Xin Zhai, Hanwen Du, Junchen Fu, Wenhao Deng, Yongxin Ni, and Chunxiao Li. Skillaudit: From fixed-suite benchmarking to skill-centered assessment, 2026. URL https://arxiv.org/abs/2606.22613.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents, 2024. URL https://arxiv.org/abs/2307.13854.

Appendix Roadmap

Appendix A documents benchmark construction and provides two complete examples. Appendix B specifies how source edits are measured and consolidated. Appendix C reports the resulting experience library, Appendix D provides supplementary experimental evidence, and Appendix E gives a complete transfer case. Appendix F provides the optimization prompt templates.

A Benchmark Construction Details

A.1 Collection and Filtering

We collect skill packages from 200 license-audited public sources. Each package contains a `SKILL.md` file and may include scripts, references, or assets. We deduplicate packages by instruction content, retaining the more authoritative and complete copy, and exclude malformed, routing-only, harmful, or sensitive packages. We also exclude skills whose main workflow cannot be reproduced offline without external credentials, remote resources, or specialized hardware. This process yields 20,579 eligible candidates.

Table 3: Skill collection and selection pipeline.

Stage	Operation	Count
Sources	License-audited public source directories	200
Collection	Packages after within-repository content deduplication	34,231
Filtering	Eligible candidates after deduplication and filtering	20,579
Auditing	Candidates reviewed for benchmark feasibility	514
Feasibility	Candidates retained after auditing	388
Selection	Three skills from each of 25 structural cells	75

A.2 Taxonomy Definition and Annotation

Each candidate receives one label on artifact form (F), verification support (V), execution dependency (E), and control flow (C). F is derived from package contents; an LLM annotates V, E, and C from the instructions and package structure. E follows the highest interface required on the main workflow path, and C distinguishes direct generation, predefined multi-step procedures, and feedback-adaptive workflows.

Table 4: SkillPrism taxonomy and marginal counts in the classified candidate pool and final benchmark.

Dimension	Category	Decision criterion	Pool	Final
Artifact form (F)	F0 Instruction-only	Instructions plus metadata and other nonfunctional files.	12,007	21
	F1 Script-backed	Auxiliary resources consist only of executable scripts.	689	12
	F2 Reference-backed	Auxiliary resources consist only of documents for consultation.	1,532	24
	F3 Asset-backed	Auxiliary resources consist only of templates, media, or example data to copy, fill, or edit.	2,159	3
	F4 Mixed	At least two types of auxiliary resources.	4,192	15
Verification support (V)	V0 No explicit support	Supplies general guidance without explicit acceptance criteria or executable checks.	10,306	24
	V1 Static criteria	Provides fields, formats, acceptance items, or checklists that can be checked manually.	9,463	39
	V2 Executable checks	Includes or explicitly requires tests, validators, or other executable checks of deliverables.	810	12
Execution dependency (E)	E0 Context-only processing	Processes information and produces the result within the available context.	9,888	12
	E1 File operations	Requires reading or writing files on disk.	2,698	15
	E2 Command execution	Requires commands or processes whose execution results are necessary for completion.	4,072	21
	E3 Service calls	Requires operations through service interfaces, including local services.	3,921	27
Control flow (C)	C0 Single generation	Produces the deliverable directly, with at most one lookup before generation.	12,478	15
	C1 Predefined multi-step	Uses dependent actions with predetermined steps, branches, and retry rules.	7,608	48
	C2 Feedback-adaptive	Uses open-ended feedback to determine subsequent action paths.	493	12

A.3 Stratified Selection and Split

Candidates are ranked within each joint $F \times V \times E \times C$ cell using

$$s(p) = \tfrac{1}{2}\big(\operatorname{mean} Q(p) + \operatorname{mean} B(p)\big), \tag{4}$$

where Q measures completeness, specificity, reusability, and clarity, and B measures task generatability, checkability, and offline reproducibility. A feasibility audit retains 388 of 514 reviewed candidates. We then select 25 cells spanning all taxonomy values and keep three skills per cell, yielding 75 skills across 13 domains.

Table 5 reports the complete selected-cell roster. Daggers mark held-out skills. After task construction and selection, a sequential balancing procedure forms 50 source and 25 held-out skills by minimizing distributional imbalance over domain, F/E/V/C labels, and evaluator type. Evaluation outcomes are not used in either skill selection or splitting.

Table 5: The 25 selected taxonomy cells and their three skills. A dagger marks a held-out skill; all others belong to the 50-skill source split.

Cell	Skills
F0/E0/V0/C0	accessibility-fundamentals†, cads-ads-optimize, routeros-mndp†
F0/E0/V1/C0	smart-model-switching†, jtbd-extractor, ad-copy-google-ads
F0/E1/V1/C1	changelog-generator, cads-ads-google, entity-compliance
F0/E2/V1/C1	binary-re-triage†, lease-abstraction, memory-optimization-workflow
F0/E2/V2/C2	adding-tests, document-skills-pptx, sys-guard-linux-remediator
F0/E3/V0/C1	cads-ads-competitor†, fachanwalt-bau-architektenrecht-bauablauf-vbg, gplay-recovery†
F0/E3/V1/C1	connection-pool-tuner, reviewing-b2b-service-contract, cads-ads-launch†
F1/E2/V2/C1	temper-skills†, xlsx, suricata-offline-evejson
F1/E3/V0/C0	find-arbitrage-opps, email-intelligence, ops-journal
F1/E3/V1/C1	db-migration, ceaser-send†, suisec
F1/E3/V2/C1	hn-daily, webnovel-serial-pipeline†, deploying-kafka-k8s
F2/E0/V0/C0	model-strategy, mimeo-kaiming-he†, mimeo-confucius†
F2/E1/V0/C2	bar-exam†, calorie-tracker†, plan-interview
F2/E1/V1/C1	router, topic-evidence-mapper, introduction-section-writer
F2/E1/V1/C2	expat-companion, clinical-question-clarifier, domain-modeling†
F2/E2/V1/C1	betting, investor-reviewer†, risk-prioritiser
F2/E3/V0/C1	recommendation-system, scienceworld-tool-user, perplobster
F2/E3/V0/C2	senado-federal, amazon, webshop-option-selector
F2/E3/V1/C1	frontline-skeptic-reviewer, webshop-product-selector, pricewin-booking-assistant
F3/E0/V1/C0	crypto-misuse, mobile-security†, unreal-engine-cpp-pro†
F4/E1/V1/C1	claude-code-skillforge, medication-reconciliation†, deep-reading†
F4/E2/V0/C1	channel-economics†, eval-skills, product-manager-toolkit†
F4/E2/V1/C1	statistical-power†, decision-matrix, agenthub
F4/E2/V2/C1	agent-designer, skills-factory, experimental-design
F4/E3/V1/C1	ghost-catalog, evidenceops†, clawsec-suite†

A.4 Task and Evaluator Construction

We extract 441 candidate capability claims from the selected skills and retain 395 after removing claims answerable from the request alone, direct outputs of bundled scripts, or general model knowledge. Each task targets one retained claim and specifies the goal, constraints, inputs, and deliverable without prescribing the execution procedure. Inputs are adapted from public documents, code, and de-identified data when available, and synthesized otherwise. They are obtained independently of the evaluated skill and remain identical across evaluation conditions. Service-dependent tasks use local substitutes that preserve the required interaction.

Table 6: Candidate and final task composition. Each skill has eight candidate tasks and contributes four tasks to the final benchmark.

Statistic	Candidate pool	Final set
Tasks	600	300
Tasks per skill	8	4
Covered capability claims	395	300
Rule-based evaluators	282	141
Rubric-based evaluators	318	159
Real-source materials	188	93
Synthetic materials	412	207

Each candidate is assigned a design complexity tier using the operational criteria in Table 7. These labels describe the constructed task rather than difficulty inferred from model performance.

Reference solutions and evaluators are derived from the task requirements and inputs in a separate context that excludes the evaluated skill. Rule-based evaluators check outputs against computable criteria, whereas weighted rubrics assess open-ended deliverables. We validate them on reference

Table 7: Operational criteria for task complexity. Assertions are independently verifiable requirements; mechanisms are distinct sources of task difficulty.

Tier	Required work	Complexity indicators
L1	One artifact	At most five assertions and one mechanism
L2	Multiple inputs or intermediate artifacts	Six–twelve assertions and two mechanisms
L3	At least three stages or twelve assertions	An incomplete or adversarial premise and at least three mechanisms

outputs, deliberately defective outputs, and semantically equivalent variants. Paired pilot runs check solvability and calibrate task difficulty through at most two rounds of material or constraint adjustments while preserving the target claim. For benchmark evaluation, the same evaluators are used across conditions without access to the condition label.

For each skill, we enumerate all $\binom{8}{4} = 70$ four-task subsets and select the subset maximizing

$$S(T) = C_{\text{claim}}(T) + C_{\text{tier}}(T) + \tfrac{1}{2}C_{\text{eval}}(T) - 2R(T), \tag{5}$$

where the three coverage terms count represented capability claims, complexity tiers, and evaluator types, and $R(T)$ is the maximum pairwise similarity between task requests. Selection uses only these construction attributes, not pass rates, skill gains, runtime outcomes, or agent steps. The selected tasks cover four distinct claims and all three tiers for every skill, yielding the composition in Table 6.

A.5 Complete Benchmark Examples

The following examples show the skill, task, expected answer, and scoring rule for one rule-based and one rubric-based benchmark instance.

Example 1: Rule-based betting analysis

Skill Analyzes supplied odds by converting formats, removing bookmaker margin, measuring edge and expected value, and computing Kelly bet sizes. Complete bookmaker markets must be de-vigged before comparison with prediction-market prices; the skill does not retrieve live data.

Task Given a four-event worklist, mixed-format bookmaker odds, and quoted market prices, return ordered JSON records containing the requested side, fair and quoted probabilities, expected value per dollar, and nonnegative Kelly stake. Use null when fair probability cannot be identified.

Expected answer

Side	Book odds	q	Fair p	EV	Stake
MIL	−150, +130	0.55	0.579832	0.054240	0.066293
ARS	1.8, 3.6, 4.5	0.48	0.526316	0.096491	0.089069
JOK	−200 only	0.68	null	null	0
CHI	−110, −110	0.58	0.500000	−0.137931	0

Scoring The evaluator checks JSON fields, order, and values. For complete markets, implied probabilities are normalized and $\mathrm{EV} = p/q - 1$, $f = \max\{0, (p-q)/(1-q)\}$. Tolerance is 1% relative or 10^{-3} absolute; half- and quarter-Kelly stakes are accepted. The incomplete JOK market requires null fair probability and EV, and zero stake.

Example 2: Rubric-based role description

Skill Applies Confucian frameworks to leadership and organizational problems. This case uses "Rectification of Names" to compare a formal title and stated responsibilities with actual work, without inferring motives or character.

Task Draft an English memo describing the current "Head of Family Doctor Studio" position for regulatory self-inspection. The duty statement assigns contracting and record filing, chronic-disease follow-up review, and service-quality sign-off. The memo must describe the current arrangement and defer decisions about the title or incumbent.

Week starting	Triage desk	Contract/follow-up	Chronic review
2026-07-13	28	2	1
2026-07-20	30	1	0
2026-07-27	27	3	1
2026-08-03	29	2	0
2026-08-10	31	1	0

Expected answer State the formal title, list all three duties, and report the weekly hours exactly. Neutrally identify the mismatch between formal responsibilities and observed work, leaving personnel and title changes undecided.

Scoring Nine criteria check the required facts, explicit comparison, neutral wording, and absence of unsupported motives or recommendations. High-, medium-, and low-severity weights are 2, 1, and 0.5, with a normalized pass threshold of 0.8. Fabricated facts or non-English main text trigger automatic failure.

B Supplementary Optimization Protocol

Runs use OpenHands with model parameters unchanged. Task failures and timeouts remain in evaluation; infrastructure failures are rerun under the same configuration. Source learning processes skills once in batches of five, rerunning candidate edits with the original execution model. The editor uses DeepSeek-V4-Flash-0731 for attribution, experience induction, and target rewriting. Held-out rubric evaluation uses Claude Sonnet 4.6 as the primary judge. General rewriting supplies the original `SKILL.md` with skill-authoring guidance on discovery, conciseness, instruction specificity, progressive disclosure, descriptions, and workflows.

B.1 SOURCE EXPERIENCE MEASUREMENT

Each candidate tests one local editing hypothesis with explicit preservation constraints. It is applied to an independent copy of the original skill, and the associated task is rerun with the same execution model, agent harness, environment, and evaluator. Infrastructure-affected reruns are excluded from experience evidence.

Activation edits revise the description and are labeled REPAIRED when the agent subsequently accesses the skill, or UNCHANGED otherwise. Task scores and pass decisions are recorded separately. For execution edits, let a' indicate skill access in the edited run, s, s' denote the original and edited evaluator scores, and y' denote the edited pass decision; all original cases failed. We assign

$$e = \begin{cases} \textsc{Unexposed}, & a' = 0, \\ \textsc{Repaired}, & a' = 1,\ y' = 1, \\ \textsc{Improved}, & a' = 1,\ y' = 0,\ s' > s, \\ \textsc{Worsened}, & a' = 1,\ y' = 0,\ s' < s, \\ \textsc{Unchanged}, & \text{otherwise.} \end{cases} \tag{6}$$

Restored access, task repair, and score improvement provide positive evidence for their respective editing mechanisms. Worsened outcomes can restrict applicability or preservation conditions.

B.2 EXPERIENCE LIBRARY UPDATE

Measured experiences are consolidated after each source batch into (WHEN, EDIT, PRESERVE) strategies. WHEN must be observable from the target skill and its static attributes, EDIT specifies one reusable change to an instruction or workflow relation, and PRESERVE records capabilities, interfaces, facts, and successful behavior that must remain intact. Each strategy also retains taxonomy cues. For each candidate edit, we retrieve strategies from the same editing stage by taxonomy and domain matches and token overlap between the candidate description and strategy text. The union of these candidates provides context for the batch update. A new strategy requires positive evidence. Compatible evidence is merged into an existing strategy; regression evidence narrows its condition or preservation boundary; unrelated strategies remain unchanged.

After the final update, the editor uses the target package and taxonomy attributes to assess the strategies' WHEN conditions and rewrites the skill using applicable strategies. The resulting `SKILL.md` is shared across evaluation LLMs.

C EXPERIENCE LIBRARY ANALYSIS

C.1 SOURCE EDIT OUTCOMES AND LIBRARY ACCUMULATION

We sample 307 trajectories from 414 eligible failed source executions for diagnosis and measure 144 candidate edits. The 92 REPAIRED outcomes comprise 29 activation edits that restore skill access and 63 execution edits that turn task failure into success. A further ten execution edits improve scores, giving 102 positive measurements. The remaining edits comprise 35 unchanged, six worsened, one unexposed, and no infrastructure-affected reruns (Figure 5(a)).

The 46 source skills with diagnosed failures are processed in ten batches. Cumulative measurements reach 144, while evidence merging yields a library of 28 strategies rather than one strategy per edit (Figure 5(b)–(c)).

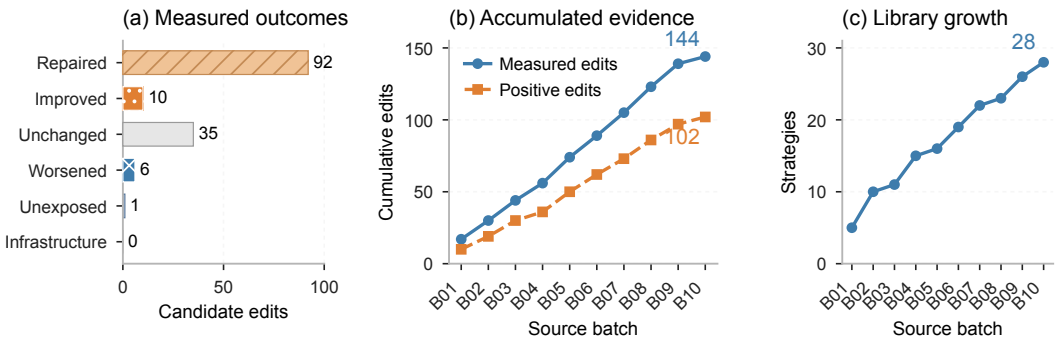

Figure 5: **Source edits provide measured evidence for an experience library.** **(a)** Outcome counts over 144 candidate edits; REPAIRED includes 29 restored-access and 63 task-repair outcomes. **(b)** Cumulative measured and positive edits across ten source batches. **(c)** The strategy library grows to 28 entries as compatible evidence is merged.

C.2 LEARNED STRATEGIES

Table 8 summarizes the final library.

Table 8: The 28 strategies learned from measured source edits. Strategy descriptions state the reusable editing mechanism; +, −, and Skills give positive evidence, negative boundary evidence, and distinct supporting source skills.

ID	Learned editing mechanism	+	−	Skills
Activation				
P01	Expose concrete, body-supported capabilities in narrow or abstract descriptions.	29	0	29
Execution				
P02	Stop probing once the outcome is determined or documented access is exhausted.	5	0	4
P03	Bind identifiers to the user-facing logical key rather than repository identity.	2	0	2
P04	State the prose decision or review artifact contract at the writing step.	7	1	5
P05	Use the environment-provided local interface as a bounded service fallback.	3	0	3
P06	Bind item sets, order, labels, filtering, and identity to source selection criteria.	10	2	9
P07	Invalidate every stale layer affected by an invalidating action.	1	0	1
P08	Derive conclusions from authoritative rules and explicitly resolve conflicting inputs.	6	0	5
P09	Attach the governing-rule checklist to clause review and require subsection findings.	1	0	1
P10	Treat a monolithic input as source material when packaging a document bundle.	1	0	1
P11	Preserve source values, absent/unknown semantics, attribution, and effective-source recalculation.	16	1	11
P12	Gate current recommendations on documented feasibility and readiness constraints.	2	0	2
P13	Use a canonical leading verb and mirror the requester's process framing.	1	0	1
P14	Preserve procedural structure when prose-only output is requested.	1	0	1
P15	Resolve numeric configuration values against declared defaults and limits.	1	0	1
P16	Commit repeated-work plans to one reusable query foundation and routing course.	1	0	1
P17	Prevent duplication across generated item sets and complementary fields.	2	0	2
P18	Use supporting tools for bounded post-production verification, with fallback.	1	0	1
P19	Enforce prohibited-content constraints during output construction.	1	0	1
P20	Encode implementation-level parameter invariants in tool and API calls.	1	0	1
P21	Gate ranking and reporting on the official evaluation contract.	1	0	1
P22	Bind delegation alternatives to explicit authorization or documented fallback.	2	0	1
P23	Give randomized tests a fixed, provenance-bearing seed in the artifact.	1	0	1
P24	Preserve exact pre-change values for reversible schema and value migrations.	2	0	1
P25	Apply the documented default when the requester selects no option.	1	0	1
P26	Treat portfolio-controlled resources as one unit in adjustment recommendations.	1	0	1
P27	Bind action and verdict fields to the backend's canonical vocabulary.	1	0	1
P28	Make the documented containment escalation order non-skippable and record the action token.	1	0	1

The evidence columns report positive and negative measured edits and the number of distinct supporting source skills. The largest groups concern discoverable capabilities, source-grounded output construction, and explicit semantics for missing or state-dependent fields.

D Supplementary Experimental Results

D.1 Source Outcomes and Taxonomy Breakdown

Across the 1,200 source model–task pairs, no-skill produces 606 successes and with-skill produces 773. The paired outcomes are 222 REPAIR, 55 BREAK, 551 both-pass, and 372 both-fail cases. Thus, the pooled gain is $100 \times 167/1200 \approx 13.9$ pp, the repair rate is $222/594 \approx 37.4\%$, and the regression rate is $55/606 \approx 9.1\%$.

We record skill access when the agent explicitly invokes the skill or accesses its package through a tool call. Table 9 reports model-level outcomes for runs where the agent accesses the skill; Table 11 breaks down these results by taxonomy category.

Table 9: Observed skill access and paired outcome composition. Transition shares are computed for each model's runs where the agent accesses the skill.

			Paired outcomes (%)			
Model	Skill access (%)	n	Fail→pass	Pass→fail	Pass→pass	Fail→fail
Qwen3.8-Max	83.0	166	19.9	3.6	52.4	24.1
Gemini-3.7-Flash	88.5	177	22.6	2.8	46.9	27.7
GLM-5.2	65.5	131	21.4	6.9	41.2	30.5
DeepSeek-V4-Flash-0731	70.5	141	24.1	2.1	42.6	31.2
DeepSeek-V4-Pro-0813	68.5	137	21.2	2.2	39.4	37.2
Qwen3.7-Plus	32.5	65	30.8	1.5	35.4	32.3
Pooled	68.1	817	22.5	3.3	44.2	30.0

Table 10: Execution-cost changes for runs where the agent accesses the skill. Values are percentage changes in aggregate resource use from no-skill to with-skill within each paired outcome group.

Outcome	Cases	Input tokens (%)	Reasoning tokens (%)	Agent steps (%)	Runtime (%)
Repair	184	−28.0	−24.3	−15.6	−31.0
Regression	27	+5.9	+60.8	+5.8	+35.6
Both pass	361	+36.2	+13.0	+8.8	+7.8
Both fail	245	−7.1	+0.3	−2.0	−9.1

Model differences for mixed resources. Table 12 compares gains for mixed skills in runs where each model accesses the skill and across the same 32 tasks from eight skills. On the common task set, Gemini-3.7-Flash gains ten successful tasks, while the other models gain one to three.

Table 11: Source-skill utility by taxonomy category for runs where the agent accesses the skill. Pass, repair, and regression rates are percentages; Δ_{pass} is measured in percentage points. Repair and regression rates use no-skill failures and successes as their respective denominators.

Category	Skills	n	No-skill (%)	With-skill (%)	Δ_{pass} (pp)	Repair (%)	Regression (%)
Artifact form (F)							
F0 Instruction-only	14	261	52.1	67.8	+15.7	41.6	8.1
F1 Script-backed	9	143	59.4	80.4	+21.0	58.6	4.7
F2 Reference-backed	18	285	45.3	67.4	+22.1	45.5	6.2
F3 Asset-backed	1	22	0.0	13.6	+13.6	13.6	–
F4 Mixed	8	106	35.8	54.7	+18.9	35.3	10.5
Verification support (V)							
V0 No explicit support	14	203	48.8	70.4	+21.7	53.8	12.1
V1 Static criteria	26	474	41.8	61.6	+19.8	36.6	3.5
V2 Executable checks	10	140	65.0	78.6	+13.6	55.1	8.8
Execution dependency (E)							
E0 Context-only processing	5	96	52.1	58.3	+6.2	26.1	12.0
E1 File operations	10	178	38.8	62.4	+23.6	39.4	1.4
E2 Command execution	15	215	48.4	68.8	+20.5	47.7	8.7
E3 Service calls	20	328	50.3	70.1	+19.8	46.6	6.7
Control flow (C)							
C0 Single generation	8	150	44.7	61.3	+16.7	39.8	11.9
C1 Predefined multi-step	33	536	50.9	69.4	+18.5	43.3	5.5
C2 Feedback-adaptive	9	131	36.6	61.8	+25.2	44.6	8.3

Table 12: Mixed-skill utility by model. Tasks where the agent accesses the skill vary by model; the all-task columns use the same 32 tasks from eight mixed skills. Success counts refer to no-skill and with-skill; gains are percentage points.

	With skill access		All 32 tasks		
Model	n	Δ_{pass} (pp)	No-skill successes	With-skill successes	Δ_{pass} (pp)
Qwen3.8-Max	24	+12.5	18	21	+9.4
Gemini-3.7-Flash	25	+40.0	10	20	+31.2
GLM-5.2	12	+8.3	17	19	+6.2
DeepSeek-V4-Flash-0731	20	+15.0	16	17	+3.1
DeepSeek-V4-Pro-0813	19	+5.3	17	18	+3.1
Qwen3.7-Plus	6	+33.3	12	15	+9.4

D.2 Held-out Coverage and Skill-level Direction

Table 13 supplements the pass rates in Table 2 with gains over original skills.

Table 13: Held-out gains over original skills (pp), computed from unrounded pass rates. Each model evaluates 100 tasks with three runs per condition.

	General rewriting		SkillPrism-Opt	
Model	pass@1	pass@3	pass@1	pass@3
Qwen3.7-Plus	−3.0	−2.0	+2.7	+7.0
DeepSeek-V4-Flash-0731	−1.7	+2.0	+3.0	+2.0
GLM-5.2	−1.0	+2.0	+4.3	+6.0
Average	−1.9	+0.7	+3.3	+5.0

For a model–task cell, newly covered means zero successful original runs and at least one successful optimized run; lost coverage is the reverse. Relative to original skills, SkillPrism-Opt adds 21

covered cells and loses 6, compared with 16 additions and 14 losses under general rewriting. Across all three models, these changes correspond to 30 additional successful runs for SkillPrism-Opt and 17 fewer for general rewriting.

Table 14 reports the corresponding direction of change after aggregating each model's three runs within a skill.

Table 14: Direction of held-out skill-level changes under SkillPrism-Opt. Each model contributes 25 skills; totals count 75 skill–model combinations.

	Versus original			Versus general rewriting		
Model	Improved	Tied	Regressed	Improved	Tied	Regressed
Qwen3.7-Plus	13	6	6	15	6	4
DeepSeek-V4-Flash-0731	11	8	6	14	6	5
GLM-5.2	13	7	5	11	10	4
Total	37	21	17	40	22	13

D.3 Statistical Estimation and Uncertainty

For each held-out model and condition, we use the pass@k estimator of Chen et al. (2021):

$$\widehat{\text{pass@}k} = \frac{1}{N}\sum_{i=1}^{N}\left[1 - \frac{\binom{n-c_i}{k}}{\binom{n}{k}}\right], \qquad n = 3, \quad k \in \{1, 3\}, \tag{7}$$

where $N = 100$ tasks and c_i is the number of successful runs for task i.

Source intervals use 4,000 skill-cluster bootstrap samples, retaining all four tasks and six LLMs for each sampled skill. The pooled source gain is $+13.9$ pp with a 95% interval of $[9.3, 19.0]$.

Held-out intervals use 10,000 bootstrap samples over 25 skill clusters, retaining each skill's tasks, LLMs, conditions, and three runs. The leave-one-skill-out range removes each skill and recomputes the effect over the remaining 24.

Table 15: Cross-model held-out effects with 95% skill-cluster bootstrap intervals and leave-one-skill-out (LOO) ranges, in percentage points. R: original; G: general rewriting; O: SkillPrism-Opt; A: SkillPrism-Opt w/o Exp.

Contrast	Metric	Estimate	95% CI	LOO range
O − R	pass@1	+3.33	[1.11, 5.67]	[2.89, 3.70]
	pass@3	+5.00	[1.67, 8.33]	[4.51, 5.90]
O − G	pass@1	+5.22	[2.78, 7.67]	[4.51, 5.90]
	pass@3	+4.33	[0.33, 8.33]	[3.47, 5.21]
O − A	pass@1	+2.67	[–0.67, 5.89]	[1.97, 3.47]
	pass@3	+6.67	[2.66, 11.33]	[5.21, 7.29]

E Cross-Skill Experience Transfer Case

Table 16 traces the transfer of an editing strategy from evaluation reporting to meal logging. Both skills construct records from incomplete inputs; the transferable experience specifies how to represent missing information and distinguish estimates from observed values.

Table 16: Missing-value handling from `eval-skills` to `calorie-tracker`. Held-out outcomes evaluate the complete optimized skill.

Stage	Evidence
Source failure	`eval-skills` supports evaluation and regression reporting. On `task_006`, the report made unsupported latency claims when p95 data were missing; score 0.6667, fail.
Measured edit	Quote source metric values and explicitly mark a missing percentile as unavailable, without interpreting it as zero or normal performance. The Gemini-3.7-Flash rerun scores 1.0 and passes.
Learned strategy	The source-value strategy P11 includes this experience: retain absent/unknown values explicitly and preserve the provenance of generated fields.
Target gap	`calorie-tracker` logs meals and estimates calories. Its memory template says to observe and fill fields, but does not explicitly specify how to handle missing values or attribute estimates at the writing step.
Held-out rewrite	Added field-writing rules leave unavailable information empty, retain source attribution for estimates, and prohibit converting missing details into exact calories, zeros, or defaults.
Observed change	The task requests a meal record from notes with unknown portion sizes and recipe quantities. The inspected original run allowed another restaurant's 650 kcal value as a single-number estimate. Successful runs with the optimized skill flagged the unsupported quantities and declined to confirm exact values.
Held-out result	On `calorie-tracker/task_008` with DeepSeek-V4-Flash-0731, the three-run pass vectors are 000 for original, 001 for general rewriting, and 110 for SkillPrism-Opt.

F Prompt Templates

JSON schemas are omitted from the following templates.

F.1 Execution Experience Attribution

```
ROLE

Analyze failed trajectories in which the agent accessed the target skill,
and propose skill edits that can be validated by a rerun.

INPUT

- skill profile and complete skill package;
- failed cases with evaluator results and trajectory evidence;
- successful cases for counterevidence and preservation constraints;
- evaluator conventions and edit constraints.

ATTRIBUTION

For each case:
1. Identify the unmet task behavior.
2. Locate the earliest trajectory deviation that prevents that behavior.
3. Mark the relevant skill rule as explicit, ambiguous, absent, or conflicting.
4. Classify the primary source as skill gap, model execution deviation,
   environment failure, interaction, or uncertain.
5. Classify editability as direct, robustness-only, unlikely, or uncertain.

Propose an edit only when the evidence supports:
editable skill mechanism -> earliest deviation -> unmet behavior.

CANDIDATE EDITS

Each candidate must test one behavioral hypothesis, identify a local edit
anchor, predict an observable behavior change, and list behavior and
interfaces that must be preserved. Do not include task answers, task IDs,
fixed sample values, evaluator implementation, new tools, or unsupported
domain rules. Return no candidate when a credible skill-level intervention
cannot be established.

OUTPUT

Return only the JSON object required by the supplied schema, with concise
```

```
evidence references and no unmeasured cross-skill conclusions.
```

F.2 EXPERIENCE INDUCTION

```
# Role

Update an existing skill-editing policy library from one batch of measured
candidate edits.

# Goal

Produce compact, transferable `WHEN--EDIT--PRESERVE` policies. The result
must explain what an editor should inspect in an unseen skill, what behavioral
relation it should add or revise, and what successful behavior must remain
unchanged.

# Evidence Use

- A repair or score improvement can support a policy.
- A regression can only define a boundary or motivate narrowing.
- An unchanged or unexposed result cannot support a policy.
- Prefer policies supported by multiple skills or models.
- Keep a singleton only when its mechanism is precise, observable on an
  unseen skill, and cannot be defensibly merged with another policy.
- Taxonomy and domain are retrieval hints. They are not sufficient `when`
  conditions.

# Abstraction Test

Every emitted policy must pass all four tests:

1. Its condition is observable from the target skill, its metadata, bundled
   resources, and declared environment, without target tasks or trajectories.
2. Its edit changes one identifiable agent decision or workflow relation.
3. It excludes source skill names, task wording, judge fields, and fixed
   example values.
4. It is more specific than generic advice such as "clarify instructions",
   "add validation", or "handle edge cases".

Good mechanisms include exposing a body-supported capability in the
description, binding a governing constraint where an output is constructed,
defining absent-value semantics, adding a bounded dependency fallback, or
turning a check result into an explicit next action. These examples indicate
the desired granularity; they are not a predefined repair taxonomy.

# Updating the Retrieved Neighborhood

Each current policy in the retrieved neighborhood must either:

- appear in exactly one revision through `source_policy_ids`; or
- appear in `deleted_policy_ids`.

Use one revision to retain, narrow, or merge source policies. A revision with
no source policy creates a new policy. Preserve a source policy's wording when
the new batch provides no evidence that changes it. Merge policies only when
one observable condition and one editing mechanism cover their evidence.

`evidence_ids` and `boundary_evidence_ids` contain candidate IDs from the
current batch, never case IDs. Do not repeat evidence already stored in a
source policy: the program carries that evidence forward automatically.
`evidence_ids` may contain only candidates with measured repair or
improvement. `boundary_evidence_ids` may contain only candidates with measured
regressions. Neutral and unexposed candidates should normally be rejected.
An admitted candidate requires positive evidence. A narrowed candidate may
instead be a measured regression that supplies a boundary for an existing
supported policy; such a candidate cannot be the sole basis of a new policy.

Return JSON only and follow the supplied schema exactly.
```

F.3 SKILL BODY REFINEMENT

```
# Role

You are rewriting one held-out agent skill using a library of editing
experiences learned from measured before/after experiments on other skills.

You receive the target skill package, its static SCALES taxonomy, and the
complete experience library. You do not receive target tasks, trajectories,
```

```
evaluator rules, scores, or target-specific diagnoses.

# Objective

Produce one complete, coherent rewrite of `SKILL.md`. Use the experience
library as empirical editing knowledge: identify analogous weaknesses in the
target skill, transfer the supported mechanisms, and respect the harmful
boundaries recorded by the experiments.

This is not retrieval by exact domain, field name, or taxonomy cell. An
experience may transfer when the target has the same underlying instruction or
execution problem even if its task and domain differ.

# Interpreting the Experiences

Each experience contains:

- observable conditions under which an edit may help;
- a failure mechanism;
- a behavioral objective;
- an editing strategy;
- behavior and interfaces that must be preserved;
- harmful generalizations to avoid;
- measured positive evidence and, when available, regression boundaries.

Treat the concrete source domain as an example, not as the scope of the
experience. Abstract its mechanism before applying it. For example, a lesson
about checking inputs before a financial calculation may transfer to another
deterministic operation with required inputs, but its betting-specific formulas
and output values must not transfer.

Do not apply an experience merely because the target shares a taxonomy label.
The target must contain a visible analogous mechanism. Conversely, do not
reject an experience merely because the target uses different entities,
commands, artifacts, or domain terminology.

# Rewrite Procedure

1. Read the complete target package and identify the skill's purpose, trigger,
   workflow, dependencies, decision points, output contracts, and validation
   behavior.
2. Identify weaknesses visible from the skill text itself, such as:
   - an operation whose prerequisites are implicit;
   - an external or local dependency with no bounded failure path;
   - a global requirement that is absent from the local step where it matters;
   - an output contract that is incomplete at delivery time;
   - a validator whose input scope or decision criteria are ambiguous;
   - evidence of different strengths being treated as equivalent;
   - a deterministic helper being followed without a prior routing decision.
3. Map each diagnosed weakness to one or more measured experiences at the
   mechanism level. Cite exact target `SKILL.md` spans that support the mapping.
4. Rewrite the complete `SKILL.md` so the selected mechanisms are integrated
   into the natural location in the workflow rather than appended as detached
   advice.
5. Review the rewritten document for internal consistency, preservation of
   existing behavior, and accidental transfer of source-specific content.

# Rewrite Constraints

- Return the complete rewritten `SKILL.md`, not a patch.
- Preserve the YAML frontmatter exactly.
- Preserve the skill's identity, scope, supported capabilities, paths,
  commands, scripts, formulas, thresholds, schemas, allowed values, exceptions,
  and domain facts unless reorganizing their presentation without changing
  their meaning.
- Do not add tools, dependencies, capabilities, facts, or domain rules absent
  from the target package.
- Do not copy induction-skill names, entities, paths, field names, literal
  values, examples, or evaluator language.
- Do not assume hidden target failures. Diagnose only weaknesses visible in the
  supplied package.
- Integrate changes into the existing document structure. Remove duplication
  introduced by the rewrite and keep the result concise enough to execute.
- Do not force all experiences into the target. Every reported change must cite
  at least one relevant experience and visible target evidence.
- Apply regression-derived `avoid` and `preserve` boundaries whenever an
  experience is used.

# Output

Return one JSON object that validates exactly against the supplied schema.
Return JSON only, without Markdown or commentary outside the JSON object.
```

F.4 SKILL DESCRIPTION REFINEMENT

```
# Role

You rewrite the YAML frontmatter description of one held-out agent skill using
conditional activation policies learned from measured before/after experiments
on other skills.

You receive the target skill's SCALES taxonomy, current description, and
complete `SKILL.md`. You do not receive held-out tasks, trajectories, evaluator
rules, scores, or target-specific diagnoses.


# Objective

Improve whether an agent recognizes the skill as relevant and reads
`SKILL.md`. Rewrite only the YAML `description`. Preserve the skill name, body,
resources, scope, and exclusion boundaries.


# Interpreting the Policies

Each experience describes:

- an observed activation failure;
- target-visible conditions that indicate the same semantic gap;
- an editing decision that repaired activation on other skills;
- cues and scope boundaries that the edit must preserve.

Treat source examples as evidence for the policy, not as an exhaustive
applicability checklist. First abstract the underlying activation mechanism,
then inspect whether the target description has an analogous omission.

Do not reject an experience merely because the target uses different domains,
entities, artifacts, terminology, or workflow organization. Taxonomy and
domain guide inspection but do not determine applicability by themselves.
Apply an experience only when the target body supports the capability,
operation, input, deliverable, lifecycle stage, or decision vocabulary that
would be surfaced.


# Rewrite Procedure

1. Read the current description and the complete skill body.
2. Identify the body-supported activation surface:
   - user intents and requested actions;
   - accepted input or source-material forms;
   - produced artifacts, decisions, or outcomes;
   - alternate modes and lifecycle stages;
   - common user-facing terms corresponding to expert or acronym vocabulary.
3. Compare that surface with the current description. Look for analogous gaps
   described by the activation experiences, even when their concrete examples
   differ from the target.
4. Select every experience whose mechanism is supported by visible target
   evidence. Prefer complementary experiences when they expose different
   trigger dimensions.
5. Rewrite the description as a concise activation surface that preserves
   existing cues while adding the missing intent, input, operation,
   deliverable, mode, or decision vocabulary.
6. Review the description for unsupported scope expansion and accidental loss
   of existing triggers or exclusions.


# Editing Constraints

- Edit only the YAML `description`; do not edit the skill body or auxiliary
  files.
- Preserve every existing supported capability, precise trigger, and explicit
  exclusion.
- Add only capabilities demonstrably supported by the supplied package.
- Express what user requests should invoke the skill, not the full procedure
  for completing those requests.
- Do not copy step-by-step workflows, formulas, thresholds, command sequences,
  schemas, or validation checklists into the description.
- Do not copy source-skill names, entities, task wording, filenames, dates, or
  examples from policy evidence.
- Do not mention SCALES, benchmarks, experiments, activation, or the policy
  library.
- Do not use generic phrases such as "use for related tasks."
- Prefer a compact capability statement followed by concrete trigger
  vocabulary. Avoid an exhaustive prose inventory.
- If no policy has an analogous target-side mechanism after this semantic
```

```
  inspection, preserve the current description unchanged.


# Evidence

For each selected experience, identify text from the target description or
body that supports the transfer. The evidence may locate the relevant
capability without literally stating the missing trigger.


# Output

Return one JSON object conforming exactly to the supplied schema. Output JSON
only.
```