SKILLOPT: Executive Strategy for Self-Evolving Agent Skills

Yifan Yang[1,4,*,†] **Ziyang Gong**[1,*,§] **Weiquan Huang**[2,*,§] **Qihao Yang**[1,*,§]
Ziwei Zhou[3,*,§] **Zisu Huang**[3,*,§] **Yan Li**[1,§] **Xuemei Gao**[4] **Qi Dai**[4]
Bei Liu[4] **Kai Qiu**[4] **Yuqing Yang**[4] **Dongdong Chen**[4] **Xue Yang**[1,‡] **Chong Luo**[4]

[1]Shanghai Jiao Tong University [2]Tongji University [3]Fudan University [4]Microsoft

yifanyang@microsoft.com, yangxue2019-sjtu@sjtu.edu.cn

Code: https://aka.ms/SkillOpt

Abstract

Agent skills today are hand-crafted, generated one-shot, or evolved through loosely controlled self-revision—none of which behaves like a deep-learning optimizer for the skill, and none of which reliably improves over its starting point under feedback. We argue the skill should instead be *trained* as the external state of a frozen agent, with the same discipline that makes weight-space optimization reproducible. SKILLOPT is, to our knowledge, the first systematic *controllable* text-space optimizer for agent skills: a separate optimizer model turns scored rollouts into bounded add/delete/replace edits on a single skill document, and an edit is accepted only when it strictly improves a held-out validation score. A textual learning-rate budget, rejected-edit buffer, and epoch-wise slow/meta update make skill training stable while adding zero inference-time model calls at deployment. Across six benchmarks, seven target models, and three execution harnesses (direct chat, Codex, Claude Code), SKILLOPT is best or tied on **all 52 evaluated (model, benchmark, harness) cells** and beats every per-cell competitor among human, one-shot LLM, Trace2Skill, TextGrad, GEPA, and EvoSkill skills. On GPT–5.5 it lifts the average no-skill accuracy by $+\mathbf{23.5}$ points in direct chat, by $+24.8$ inside the Codex agentic loop, and by $+19.1$ inside Claude Code. Transfer experiments further show that optimized skill artifacts retain value when moved across model scales, between Codex and Claude Code execution environments, and to a nearby math benchmark without further optimization.

1 Introduction

Frontier language models are increasingly deployed as agents, from single-prompt callers to multi-step execution harnesses with tools, files, and verifiers [1–4]. In such settings, domain adaptation is no longer only about model weights or prompts: it also requires improving the *procedures* by which the agent gathers evidence, calls tools, follows domain conventions, and formats outputs [5, 6]. Agent skills provide a natural interface for this procedural adaptation [7, 8]: a skill is a portable natural-language artifact that packages procedures, domain heuristics, tool policies, output constraints, and failure modes, letting a frozen agent adapt through external text.

If the recurring object of adaptation is the agent's procedure, the skill document itself should be trainable. Yet weight adaptation is often unavailable for closed frontier models and expensive for open ones, while manually written or one-shot skills are brittle under a target domain or harness. Recent systems convert execution experience into reusable textual artifacts—distilling trajectory lessons,

*Equal contribution. †Project leader. ‡Corresponding author. §Work done during an internship at MSRA.

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

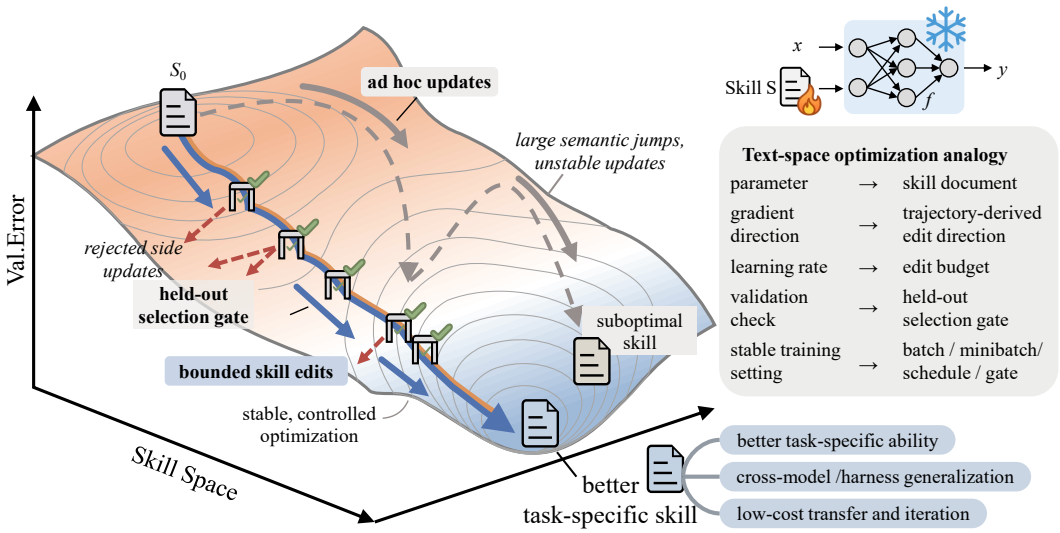

Figure 1: Overview of SKILLOPT. The target model executes tasks with a current skill, an additional frontier optimizer model converts trajectories into bounded add/delete/replace skill edits, and a held-out gate accepts only edits that improve validation performance. Accepted edits are exported as a reusable skill artifact, while rejected edits become negative feedback for later updates.

refining skill folders via failure analysis, building domain-specific skill libraries, or optimizing prompts from trajectory feedback [9–13]—but leave open a more basic question: if skills are the adaptation layer, how should they be optimized? Our key idea is to treat skill editing as a controllable domain-adaptation process, with the skill document as the external state, an additional frontier model as the optimizer, and training-style controls over evidence, step size, validation, and update direction.

We introduce SKILLOPT, a text-space optimizer for agent skills. Given a target domain, an initial skill, and the model being adapted, SKILLOPT repeatedly samples trajectory batches, analyzes successes and failures, and asks a frontier optimizer model to propose structured add/delete/replace edits. It then aggregates and ranks candidate edits under a textual learning-rate budget, applies a bounded update to the skill document, and evaluates the candidate skill on a held-out selection split before accepting it. Rejected edits are retained as negative feedback, while the epoch-wise slow/meta update preserves longer-horizon regularities. Figure 1 gives a schematic view of this loop. The deployed output is a compact `best_skill.md` file of roughly 300–2,000 tokens, with the adapted model and execution harness remaining fixed.

The deep-learning analogy is a design principle rather than a claim of a new optimization paradigm, but it is operational rather than decorative. Rollout and reflection batch sizes control the noise in the evidence used for each edit; the textual learning rate and schedule control how far one skill version is allowed to move from the previous one; the held-out gate plays the role of validation; and the epoch-wise slow/meta update acts like a momentum term, carrying stable editing directions across epochs. This stability is crucial: if consecutive skill revisions move too far or in inconsistent directions, rejected edits and previous accepted edits no longer provide a meaningful optimization history. With bounded, validation-gated updates, each revision remains close enough to the last one that later optimizer calls can learn from what helped, what failed, and what should be preserved.

We conduct, to our knowledge, the first systematic study of skill optimization as a domain-adaptation training method for frontier agents. We evaluate SKILLOPT on six benchmarks covering QA, spreadsheets, documents, math, and embodied decision making, across seven target models from frontier-scale GPT to small-scale Qwen, and under three execution modes (direct chat, Codex harness, Claude Code harness). Out of 52 evaluated (model, benchmark, harness) cells, SKILLOPT is the best or tied-best measured method on all 52. With GPT–5.5 in direct chat, it lifts SearchQA from 77.7 to 87.3, SpreadsheetBench from 41.8 to 80.7, OfficeQA from 33.1 to 72.1, DocVQA from 78.8 to 91.2, LiveMathematicianBench from 37.6 to 66.9, and ALFWorld from 83.6 to 95.5 (a $+23.5$ point average gain over no skill), and it also beats the strongest *per-cell* baseline drawn from human-written, one-shot LLM, Trace2Skill, TextGrad, GEPA, and EvoSkill skills by $+5.4$ points on average. The

same optimization interface is effective inside Codex-style and Claude Code-style execution loops, lifting GPT–5.5 by $+24.8$ and $+19.1$ points over no skill respectively, and outperforming EvoSkill by $+14.0$ and $+3.2$ points.

The learned artifacts also transfer beyond the exact training setting. A SpreadsheetBench skill trained on GPT–5.4 improves every smaller GPT variant we test; a Codex-trained spreadsheet skill transfers to Claude Code with a $+59.7$ point gain; and an OlympiadBench skill yields positive gains on Omni-MATH [14]. These transfer results are important for the paper's application value: a skill can be optimized once, audited as text, and reused across related models, harnesses, or tasks without changing model weights. Our ablations explain why this works. Bounded textual learning outperforms uncontrolled rewriting, held-out gating prevents harmful proposals from accumulating, the rejected-step buffer converts failed edits into negative feedback, and the epoch-wise slow/meta update improves long-horizon refinement without bloating the deployed skill. Finally, per-benchmark case studies show that the learned skills remain compact (300–$2{,}000$ tokens after only 1–4 accepted edits), inspectable, and procedural rather than instance-specific.

Our contributions are as follows:

- We formulate agent-skill learning as optimization over an external natural-language state and introduce SKILLOPT, a harness-agnostic optimizer with rollout batches, reflection minibatches, add/delete/replace edits, textual learning rates, schedules, held-out acceptance, rejected-edit buffers, and epoch-wise slow/meta update.
- We provide a broad empirical study across six benchmarks, seven target models, and three execution harnesses, showing that SKILLOPT is best or tied-best on 52 of 52 cells and outperforms no-skill, human-skill, one-shot LLM-skill, prompt-optimization (TextGrad, GEPA), and skill-evolution (Trace2Skill, EvoSkill) baselines under every model.
- We validate the optimization design through component ablations and three forms of transfer (cross-model, cross-harness, cross-benchmark), showing that the exported skill artifact is compact, reusable, and deployable without model-weight updates.

2 Related Work

Prompt auto tuning and agent-configuration search. ProTeGi [15] formulates prompt optimization as textual gradient descent: natural-language critiques of minibatch failures act as gradients for editing the prompt, and beam search keeps the best-scoring candidates. TextGrad extends such textual feedback to computation graphs [16], and GEPA demonstrates that trajectory feedback can guide reflective prompt evolution and outperform reinforcement learning on several language-agent tasks [13]. ABSTRAL and EvoTest extend this idea from single prompts to multi-agent design documents and test-time agentic system evolution without gradients or fine-tuning [17, 18], and AgentOptimizer [19] trains agents offline by treating their functions as learnable weights. By treating language artifacts as optimizable objects, these methods can directly exploit execution feedback, but they mainly target prompts, system designs, tool sets, or full configurations rather than reusable domain adaptation. SKILLOPT builds on this execute–critique–edit loop but instead optimizes a persistent skill document that can be trained, validated, exported, and reused with the adapted model, applying language-level controllability to a stable procedural skill state. Unlike ProTeGi, which regenerates candidate prompts and ranks a beam by training score, SKILLOPT refines a single skill through bounded, scheduled edits that must improve a held-out selection score; Table 1 and Appendix G compare the two directly.

Skill construction and skill evolution. SkillsBench and the SoK on agentic skills frame skills as reusable procedural knowledge, covering tool policies, applicability conditions, execution routines, and supporting resources [7, 8]. Prior systems construct such skills from lifelong experience, trajectory lessons, skill knowledge bases, or heterogeneous domain resources [9, 12, 20–22], and further refine them through failure analysis, creation-evaluation-revision loops, co-evolving generators and verifiers, collective updates, or reinforcement learning [10, 11, 23–29]. While these works emphasize skill discovery, repository growth, sharing, evolutionary search, or policy optimization, SKILLOPT studies a narrower problem: how to train one compact domain skill with deep-learning-style controls such as trajectory batches, reflection minibatches, textual learning rates, validation gates, rejected-edit buffers, and slow/meta updates. This yields a controlled and auditable procedure for producing a portable `best_skill.md` without changing model weights.

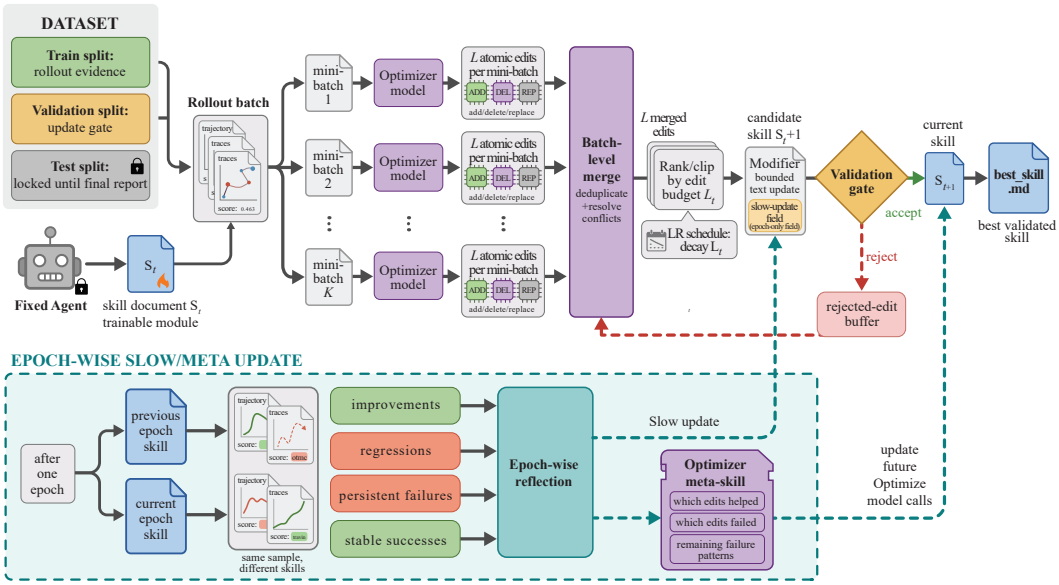

Figure 2: Pipeline of SKILLOPT. A frozen target model executes a rollout batch with the current skill; an optimizer model performs minibatch reflection over successes and failures, proposes bounded add/delete/replace edits, merges and ranks them under a scheduled edit budget, and accepts the candidate skill only through a held-out validation gate. Across epochs, the slow/meta update retains longer-horizon lessons without changing the target model.

3 Method

3.1 Problem Setup

A skill s is a natural-language policy inserted into the agent context before execution, consistent with recent work treating skills as reusable procedural knowledge for agents [7, 8]. In direct-chat benchmarks, it is prepended to the system or developer instruction; in tool-use harnesses, it becomes persistent procedural memory. We use M to denote the frozen target model whose behavior is being adapted through skill optimization. For a harness h, task x, and skill s, execution produces a trajectory τ and a scalar score r:

$$(\tau(s), r(s)) = h(M, x, s), \qquad r(s) \in [0, 1]. \tag{1}$$

Given train, selection, and test splits $D_{\text{tr}}, D_{\text{sel}}, D_{\text{test}}$, SKILLOPT uses D_{tr} to generate a set of candidate skills $\mathcal{C}(D_{\text{tr}})$, selects the best skill on D_{sel}, and reports the final performance on D_{test}:

$$s^\star_{\text{sel}} = \arg\max_{s \in \mathcal{C}(D_{\text{tr}})} \frac{1}{|D_{\text{sel}}|} \sum_{x \in D_{\text{sel}}} r(s), \tag{2}$$

$$\text{Test}(s^\star_{\text{sel}}) = \frac{1}{|D_{\text{test}}|} \sum_{x \in D_{\text{test}}} r(s^\star_{\text{sel}}). \tag{3}$$

The training split supplies experience, the selection split gates updates, and the test split is used only for final reporting. The optimizer state contains the current skill, the best validation-gated skill, cached skill hashes, an epoch-local rejected-step buffer, and optional slow/meta-update state. Only the best accepted skill is exported as `best_skill.md`.

3.2 Forward Pass: Rollout Evidence

At each optimization step, the target model runs a rollout batch from D_{tr} with the current skill. The harness records task metadata, messages, tool calls, observations, command outputs, final answers, verifier feedback, and benchmark-specific context such as spreadsheet previews, document references, or compact execution traces. This batch is the evidence unit: small batches update quickly but noisily,

while larger batches expose more recurring patterns before the skill changes. The implementation also supports accumulation, where several rollout batches are reflected on separately and merged into one update, decoupling execution throughput from update frequency.

3.3 Backward Pass: Minibatch Reflection

The optimizer model turns trajectories into skill edits, following the broader line of trajectory-driven reflection and prompt evolution [13, 30, 31]. It first separates failures from successes and partitions each group into reflection minibatches. This matters because single trajectories often produce anecdotal fixes, while minibatches expose reusable procedural errors: the agent consistently searches the wrong source, writes an answer in the wrong format, or fails to verify a tool result. Failure minibatches propose missing or corrective rules; success minibatches preserve behaviors that already work. Each reflection returns structured add/delete/replace edits, or in rewrite mode a small set of rewrite suggestions.

Local proposals are merged hierarchically by first consolidating failure- and success-driven edits separately, then combining them with priority on failure corrections. This step filters duplicate, contradictory, and example-specific suggestions before the optimizer selects the final bounded update.

3.4 Bounded Text Updates

The learning-rate analogue in SKILLOPT is the edit budget L_t: the maximum number of skill edits applied at step t. After aggregation, the optimizer model ranks the merged edit pool by expected utility and clips it to the top L_t edits. This is the key difference from ad hoc prompt rewriting. Unbounded rewrites can erase useful rules, introduce incompatible instructions, or overfit to a local failure; bounded updates preserve continuity while still allowing the skill to acquire new procedures. SKILLOPT supports constant, linear, cosine, and autonomous schedules. The default cosine schedule starts with larger edits and decays toward smaller consolidation steps.

The selected edits produce a candidate skill. In patch mode, edits are localized operations such as append, insert, replace, and delete; in rewrite mode, selected suggestions condition a full skill rewrite. Step-level edits cannot overwrite the protected slow-update field, so fast local changes and slower epoch-wise consolidation remain separated.

3.5 Validation Gate and Rejected-Edit Buffer

Every candidate skill is evaluated on D_{sel} with the same frozen target model and harness. If it improves over the current selection score, it becomes the new current skill; if it also exceeds the best score so far, it becomes `best_skill.md`. Otherwise it is rejected. This gate turns reflection into propose-and-test optimization rather than unconditional self-editing, which is crucial because plausible textual diagnoses can still hurt the actual target model.

Rejected updates are still useful. The optimizer records an epoch-local buffer containing observed failure patterns and, for rejected steps, the edits that were tried and the score drop they caused. Later reflection calls in the same epoch receive this buffer, so the optimizer model can avoid repeating failed edits and focus on unresolved failures. This gives the loop negative feedback during training without adding inference-time cost.

3.6 Epoch-Wise Slow/Meta Update

Fast updates learn from the current batch; the epoch-wise slow/meta update learns from adjacent epochs. At the end of an epoch, SKILLOPT samples the same training items under the previous epoch's skill and the current skill, then groups them into improvements, regressions, persistent failures, and stable successes. The optimizer model writes a concise longitudinal guidance block into a protected slow-update field, and this candidate is still passed through the validation gate. Thus slow update captures durable domain lessons while preserving the same safety check as step-level edits.

The meta skill is optimizer-side only. It summarizes which edit patterns helped, which were rejected, and which failures persisted across epochs. This meta guidance is prepended to future optimizer prompts for reflection, merging, and ranking, but it is not shipped with the target model. The

Model	Skill source	SearchQA	Spreadsheet	OfficeQA	DocVQA	LiveMath	ALFWorld
No harness / direct chat							
GPT–5.5	No skill	77.7	41.8	33.1	78.8	37.6	83.6
	Human skill	81.8$_{+4.1}$	72.9$_{+31.1}$	66.9$_{+33.8}$	90.1$_{+11.3}$	38.4$_{+0.8}$	91.8$_{+8.2}$
	LLM skill	80.9$_{+3.2}$	43.2$_{+1.4}$	51.7$_{+18.6}$	89.6$_{+10.8}$	40.0$_{+2.4}$	93.3$_{+9.7}$
	Trace2Skill	82.4$_{+4.7}$	49.6$_{+7.8}$	65.7$_{+32.6}$	90.6$_{+11.8}$	52.0$_{+14.4}$	87.3$_{+3.7}$
	TextGrad	81.4$_{+3.7}$	41.1$_{-0.7}$	42.0$_{+8.9}$	87.2$_{+8.4}$	49.2$_{+11.6}$	82.8$_{-0.8}$
	GEPA	84.8$_{+7.1}$	73.6$_{+31.8}$	63.9$_{+30.8}$	89.1$_{+10.3}$	43.2$_{+5.6}$	85.8$_{+2.2}$
	ProTeGi	86.4$_{+8.7}$	58.2$_{+16.4}$	44.2$_{+11.1}$	90.9$_{+12.1}$	56.8$_{+19.2}$	94.8$_{+11.2}$
	SkillOpt	**87.3**$_{+9.6}$	**80.7**$_{+38.9}$	**72.1**$_{+39.0}$	**91.2**$_{+12.4}$	**66.9**$_{+29.3}$	**95.5**$_{+11.9}$
GPT–5.4	No skill	76.9	41.4	50.0	77.6	36.8	75.4
	Human skill	80.1$_{+3.2}$	59.3$_{+17.9}$	59.3$_{+9.3}$	88.5$_{+10.9}$	41.6$_{+4.8}$	74.6$_{-0.8}$
	LLM skill	79.4$_{+2.5}$	46.1$_{+4.7}$	20.4$_{-29.6}$	90.4$_{+12.8}$	36.8$_{+0.0}$	83.6$_{+8.2}$
	Trace2Skill	81.2$_{+4.3}$	53.2$_{+11.8}$	51.2$_{+1.2}$	89.3$_{+11.7}$	40.0$_{+3.2}$	73.1$_{-2.3}$
	TextGrad	82.5$_{+5.6}$	38.6$_{-2.8}$	54.7$_{+4.7}$	85.3$_{+7.7}$	36.6$_{-0.2}$	78.4$_{+3.0}$
	GEPA	82.4$_{+5.5}$	61.1$_{+19.7}$	60.4$_{+10.4}$	88.3$_{+10.7}$	41.6$_{+4.8}$	74.6$_{-0.8}$
	SkillOpt	**83.1**$_{+6.2}$	**62.5**$_{+21.1}$	**62.8**$_{+12.8}$	**91.2**$_{+13.6}$	**44.0**$_{+7.2}$	**91.0**$_{+15.6}$
GPT–5.4-mini	No skill	75.9	36.1	22.1	71.4	14.7	73.1
	Human skill	77.2$_{+1.3}$	42.9$_{+6.8}$	45.9$_{+23.8}$	85.0$_{+13.6}$	28.8$_{+14.1}$	56.7$_{-16.4}$
	LLM skill	76.1$_{+0.2}$	36.8$_{+0.7}$	36.6$_{+14.5}$	86.4$_{+15.0}$	28.0$_{+13.3}$	65.7$_{-7.4}$
	Trace2Skill	78.6$_{+2.7}$	40.7$_{+4.6}$	20.9$_{-1.2}$	88.5$_{+17.1}$	**32.8**$_{+18.1}$	82.8$_{+9.7}$
	TextGrad	77.5$_{+1.6}$	38.2$_{+2.1}$	30.0$_{+7.9}$	84.0$_{+12.6}$	27.2$_{+12.5}$	70.9$_{-2.2}$
	GEPA	79.4$_{+3.5}$	42.5$_{+6.4}$	45.3$_{+23.2}$	83.7$_{+12.3}$	27.2$_{+12.5}$	81.3$_{+8.2}$
	SkillOpt	**80.2**$_{+4.3}$	**47.5**$_{+11.4}$	**48.8**$_{+26.7}$	**90.9**$_{+19.5}$	**32.8**$_{+18.1}$	**85.8**$_{+12.7}$
GPT–5.4-nano	No skill	55.8	23.5	16.3	30.8	23.2	34.3
	Human skill	66.1$_{+10.3}$	41.8$_{+18.3}$	46.5$_{+30.2}$	73.5$_{+42.7}$	16.8$_{-6.4}$	29.9$_{-4.4}$
	LLM skill	62.4$_{+6.6}$	33.6$_{+10.1}$	12.8$_{-3.5}$	71.7$_{+40.9}$	20.0$_{-3.2}$	65.7$_{+31.4}$
	Trace2Skill	69.9$_{+14.1}$	35.4$_{+11.9}$	16.3$_{+0.0}$	77.3$_{+46.5}$	25.6$_{+2.4}$	58.2$_{+23.9}$
	TextGrad	73.4$_{+17.6}$	32.5$_{+9.0}$	38.7$_{+22.4}$	68.3$_{+37.5}$	20.8$_{-2.4}$	42.5$_{+8.2}$
	GEPA	73.2$_{+17.4}$	37.2$_{+13.7}$	45.0$_{+28.7}$	71.2$_{+40.4}$	24.8$_{+1.6}$	68.7$_{+34.4}$
	ProTeGi	72.0$_{+16.2}$	33.6$_{+10.1}$	8.1$_{-8.2}$	67.7$_{+36.9}$	25.6$_{+2.4}$	66.4$_{+32.1}$
	SkillOpt	**74.8**$_{+19.0}$	**42.5**$_{+19.0}$	**50.0**$_{+33.7}$	**80.2**$_{+49.4}$	**27.2**$_{+4.0}$	**69.4**$_{+35.1}$
GPT–5.2	No skill	71.9	38.2	34.9	73.1	20.8	68.7
	Human skill	75.8$_{+3.9}$	48.2$_{+10.0}$	55.2$_{+20.3}$	89.0$_{+15.9}$	23.2$_{+2.4}$	56.7$_{-12.0}$
	LLM skill	73.7$_{+1.8}$	38.6$_{+0.4}$	14.0$_{-20.9}$	87.7$_{+14.6}$	25.6$_{+4.8}$	68.7$_{+0.0}$
	Trace2Skill	79.2$_{+7.3}$	43.9$_{+5.7}$	43.0$_{+8.1}$	87.7$_{+14.6}$	28.8$_{+8.0}$	76.9$_{+8.2}$
	TextGrad	82.6$_{+10.7}$	32.1$_{-6.1}$	51.1$_{+16.2}$	87.2$_{+14.1}$	22.4$_{+1.6}$	76.1$_{+7.4}$
	GEPA	82.7$_{+10.8}$	54.3$_{+16.1}$	53.4$_{+18.5}$	89.5$_{+16.4}$	28.8$_{+8.0}$	82.8$_{+14.1}$
	SkillOpt	**83.1**$_{+11.2}$	**57.1**$_{+18.9}$	**56.4**$_{+21.5}$	**89.6**$_{+16.5}$	**36.0**$_{+15.2}$	**85.1**$_{+16.4}$
Qwen3.5–4B	No skill	68.1	9.3	14.5	86.9	22.4	30.6
	Human skill	66.3$_{-1.8}$	16.4$_{+7.1}$	22.7$_{+8.2}$	87.8$_{+0.9}$	18.4$_{-4.0}$	28.4$_{-2.2}$
	LLM skill	65.0$_{-3.1}$	10.4$_{+1.1}$	20.9$_{+6.4}$	88.0$_{+1.1}$	28.8$_{+6.4}$	55.2$_{+24.6}$
	Trace2Skill	68.5$_{+0.4}$	19.3$_{+10.0}$	16.3$_{+1.8}$	88.0$_{+1.1}$	27.2$_{+4.8}$	64.9$_{+34.3}$
	TextGrad	60.7$_{-7.4}$	13.9$_{+4.6}$	20.9$_{+6.4}$	85.6$_{-1.3}$	10.6$_{-11.8}$	53.7$_{+23.1}$
	GEPA	68.6$_{+0.5}$	16.9$_{+7.6}$	22.7$_{+8.2}$	85.1$_{-1.8}$	28.8$_{+6.4}$	60.4$_{+29.8}$
	SkillOpt	**71.2**$_{+3.1}$	**23.9**$_{+14.6}$	**29.7**$_{+15.2}$	**89.0**$_{+2.1}$	**52.0**$_{+29.6}$	**81.3**$_{+50.7}$
Qwen3.6–35B-A3B	No skill	72.7	38.2	45.9	87.6	31.2	59.7
	Human skill	74.1$_{+1.4}$	44.3$_{+6.1}$	41.9$_{-4.0}$	88.4$_{+0.8}$	29.6$_{-1.6}$	44.8$_{-14.9}$
	LLM skill	72.6$_{-0.1}$	42.9$_{+4.7}$	46.5$_{+0.6}$	88.4$_{+0.8}$	24.8$_{-6.4}$	60.4$_{+0.7}$
	Trace2Skill	75.4$_{+2.7}$	33.2$_{-5.0}$	32.0$_{-13.9}$	90.4$_{+2.8}$	29.6$_{-1.6}$	70.9$_{+11.2}$
	TextGrad	76.4$_{+3.7}$	22.9$_{-15.3}$	33.7$_{-12.2}$	84.5$_{-3.1}$	7.2$_{-24.0}$	67.9$_{+8.2}$
	GEPA	75.8$_{+3.1}$	45.4$_{+7.2}$	43.6$_{-2.3}$	88.0$_{+0.4}$	31.2$_{+0.0}$	73.9$_{+14.2}$
	SkillOpt	**80.3**$_{+7.6}$	**47.5**$_{+9.3}$	**47.1**$_{+1.2}$	**91.4**$_{+3.8}$	**41.6**$_{+10.4}$	**82.1**$_{+22.4}$
Codex harness							
GPT–5.5	No skill	81.8	27.5	38.3	87.2	35.2	–
	Human skill	84.1$_{+2.3}$	50.7$_{+23.2}$	40.0$_{+1.7}$	88.8$_{+1.6}$	48.8$_{+13.6}$	–
	LLM skill	83.4$_{+1.6}$	25.0$_{-2.5}$	34.3$_{-4.0}$	89.8$_{+2.6}$	45.6$_{+10.4}$	–
	EvoSkill	61.4$_{-20.4}$	67.5$_{+40.0}$	42.4$_{+4.1}$	89.3$_{+2.1}$	63.2$_{+28.0}$	–
	SkillOpt	**87.3**$_{+5.5}$	**85.0**$_{+57.5}$	**51.1**$_{+12.8}$	**92.2**$_{+5.0}$	**78.4**$_{+43.2}$	–
Claude Code harness							
GPT–5.5	No skill	81.9	22.1	57.6	86.6	40.8	–
	Human skill	83.7$_{+1.8}$	37.1$_{+15.0}$	66.3$_{+8.7}$	88.0$_{+1.4}$	44.0$_{+3.2}$	–
	LLM skill	82.4$_{+0.5}$	37.1$_{+15.0}$	56.4$_{-1.2}$	89.6$_{+3.0}$	44.8$_{+4.0}$	–
	EvoSkill	84.0$_{+2.1}$	75.0$_{+52.9}$	70.3$_{+12.7}$	87.2$_{+0.6}$	52.0$_{+11.2}$	–
	SkillOpt	**85.9**$_{+4.0}$	**80.4**$_{+58.3}$	**71.5**$_{+13.9}$	**90.1**$_{+3.5}$	**56.5**$_{+15.7}$	–

Table 1: Main results on held-out test splits. Scores are percentages; within each model–harness block, bold marks the best measured entry and underlining marks the second-best entry for each benchmark. Blue cells denote SkillOpt, and small green/red subscripts show the absolute change relative to the *No skill* row of the same model in the same harness. ProTeGi was run for GPT–5.5 and GPT–5.4-nano from the same initial skill as SkillOpt (Appendix G). We omit ALFWorld under Codex and Claude Code harnesses because ALFWorld requires persistent embodied-environment interaction. SkillOpt is the best-or-tied entry on every measured cell of the table, with positive gains over the no-skill baseline throughout.

Component	Setting	SearchQA	SpreadsheetBench	LiveMath
Learning-rate form	lr=4 (default)	**87.1**	**77.5**	**61.3**
	dynamic lr	85.8	71.8	54.0
	without lr	84.6	75.7	57.3
Rejected buffer	with rejected buffer	**87.1**	**77.5**	**61.3**
	without rejected buffer	85.5	72.9	58.9
Slow/meta update	meta skill and slow update	**87.1**	**77.5**	**61.3**
	without meta skill	85.1	75.7	58.1
	without meta skill and slow update	86.3	55.0	59.7

Table 2: Component ablations for learning-rate form, rejected buffer, and epoch-wise slow/meta update. Light-blue rows mark the default setting within each component group; the learning-rate group uses the default lr=4 setting. Bold values mark the best measured result within that group and benchmark. The without-rejected-buffer row uses the matched no-buffer ablation setting.

advantage is separation of concerns: the deployed skill remains compact and portable, while training benefits from a richer record of the editing process.

3.7 Harness-Agnostic Deployment

SKILLOPT is harness-agnostic through a lightweight adapter interface, matching the broader trend toward agents embedded in tool-use and software-execution environments [1, 2, 4]. An adapter constructs train/evaluation batches, injects the current skill into the agent context, runs the native harness, and returns scored trajectories. The same optimizer therefore works for direct QA, spreadsheet execution, document reasoning, multimodal QA, embodied environments, and Codex-style or Claude Code-style execution loops. This is the main practical advantage of treating skills as the adaptation layer: a stronger optimizer model can train a reusable skill artifact offline, and the resulting `best_skill.md` can then be deployed or tested across target models, harnesses, and nearby benchmarks without changing model weights.

4 Experiments

We evaluate SKILLOPT as a text-space optimizer for frozen agents: the target model executes each task with the current skill, while an offline optimizer edits that skill from rollout evidence. The experiments answer four questions. (i) Do optimized skills improve over no-skill, human-skill, one-shot LLM-skill, prompt-optimization (TextGrad, GEPA), and skill-evolution (Trace2Skill, EvoSkill) baselines? (ii) Does the same loop work across direct chat, Codex, and Claude Code harnesses, and across seven target models from frontier-scale GPT to small Qwen? (iii) Which optimizer controls matter? (iv) What do the learned skills look like, and at what cost?

Setting. We report each benchmark's native hard score or exact-match accuracy on held-out test splits across SearchQA [32], SpreadsheetBench [33], OfficeQA [34], DocVQA [35], LiveMathematicianBench [36] (abbreviated LiveMath in tables), and ALFWorld [37], using two model families: GPT [38] and Qwen [39, 40]. Dataset-backed runs use deterministic train/selection/test splits derived from the same dataset seed (`split_seed` $= 42$); the selection split is used *only* to accept or reject candidate skill edits, and all reported scores are computed on the disjoint held-out test split.

Default optimizer hyperparameters. Unless noted, SKILLOPT uses four epochs, rollout batch size 40, reflection minibatch size 8, textual learning rate $L_t = 4$ with cosine decay, held-out validation gating, slow update, and the optimizer-side meta skill (full configuration in Appendix E). Direct chat prepends the skill to the system prompt, while the Codex [41] and Claude Code [42] harnesses render it as a per-task `SKILL.md` and return compact execution traces to the optimizer. Additional benchmark, baseline, and optimizer-protocol details are in Appendix E; three-seed and additional-model results, imperfect-reward studies, further baselines, and cost and reproducibility details are in Appendices H–L; optimizer strength and the learned skills are analyzed in Appendix F.

Baselines. We compare against seven baselines that span the no-adaptation, hand-written, one-shot, and learning families: *no skill* (frozen target model run with the benchmark's default system

(a) Cross-model transfer

Source model	Target model	Benchmark	Baseline	Direct	Transferred
GPT–5.4	GPT–5.4	SpreadsheetBench	41.4	62.5	–
	GPT–5.4-mini		36.1	47.5	$45.5_{+9.4}$
	GPT–5.4-nano		23.5	42.5	$26.5_{+3.0}$
GPT–5.4	GPT–5.4	LiveMath	36.8	44.0	–
	GPT–5.4-mini		14.7	32.8	$19.2_{+4.5}$
	GPT–5.4-nano		23.2	27.2	$28.8_{+5.6}$

(b) Cross-harness transfer

Source harness	Target harness	Benchmark	Baseline	Direct	Transferred
Codex	Claude Code	LiveMath	40.8	56.5	$42.4_{+1.6}$
Claude Code	Codex		35.2	78.4	$48.0_{+12.8}$
Codex	Claude Code	SpreadsheetBench	22.1	80.4	$81.8_{+59.7}$
Claude Code	Codex		27.5	85.0	$71.1_{+43.6}$

(c) Cross-benchmark transfer

Source benchmark	Target benchmark	Model	Baseline	Direct	Transferred
OlympiadBench	Omni-MATH	GPT–5.4	56.6	–	$60.3_{+3.7}$
		GPT–5.4-mini	34.8	–	$36.6_{+1.8}$
		GPT–5.4-nano	38.8	–	$40.1_{+1.3}$

Table 3: Transfer of optimized skills across three axes. **(a)** *Cross-model*: a skill optimized for the source model is deployed on the target model. **(b)** *Cross-harness*: a skill trained inside the source harness is evaluated inside the target harness, all on GPT–5.5. **(c)** *Cross-benchmark*: the source benchmark skill is evaluated on the target benchmark across three target models. **Baseline** is the target's no-skill score, **Direct** is the in-domain SKILLOPT score, and **Transferred** applies the source skill without further optimization. Subscripts show the change over the target baseline. The GPT–5.4→GPT–5.4 transferred cells in (a) are marked – because source and target match (i.e. no transfer occurs); we still report the GPT–5.4 baseline and in-domain SKILLOPT score (taken from Table 1) in those rows for reference. Every remaining row in (a)–(c) is a positive transfer (no row falls below the target's no-skill baseline).

prompt), *human skill* (an expert-written skill document curated per benchmark), *one-shot LLM skill* (a single skill generated from a high-level task description by GPT–5.5 and never updated), *Trace2Skill* [9] (trajectory-level skill distillation), *TextGrad* [16] (gradient-style natural-language prompt optimization), *GEPA* [13] (Pareto reflective prompt evolution), and the harness-side competitor *EvoSkill* [10] (skill-folder evolution under failure analysis). All baselines use the same target model, the same held-out test split, and the same scorer for every benchmark, so the comparison isolates the choice of adaptation procedure rather than secondary factors such as prompt template or scoring pipeline.

4.1 Main Results

Table 1 is the main result matrix. Counting every (target model, benchmark, harness) cell as one comparison and the strongest of the no-skill, human-skill, LLM-skill, Trace2Skill, TextGrad, GEPA, and EvoSkill baselines as the per-cell competition, SKILLOPT wins or matches the best measured result on 52 **of** 52 **evaluated cells**. This dominance is uniform across model scales: SKILLOPT is best on every benchmark for GPT–5.5, GPT–5.4, GPT–5.4-mini, GPT–5.4-nano, GPT–5.2, Qwen3.5–4B, and Qwen3.6–35B-A3B in direct chat, and for GPT–5.5 under both Codex and Claude Code harnesses.

The size of the gains is also unusually large for a no-weight-update method. On GPT–5.5 direct chat, the six-benchmark average rises from 58.8 (no skill) to 82.3 (SKILLOPT), a +23.5 point absolute improvement, while the best per-cell baseline averages only 76.9, leaving SKILLOPT +5.4 points clear of an oracle baseline that picks the best of six competing methods per cell. Per-benchmark deltas over no skill range from +9.6 on SearchQA, where the no-skill model is already near ceiling, to +38.9 on SpreadsheetBench and +39.0 on OfficeQA, where strict procedural and answer-format requirements expose the limits of zero-shot frontier models. Procedural benchmarks see the largest

improvements: SpreadsheetBench $41.8 \rightarrow 80.7$, OfficeQA $33.1 \rightarrow 72.1$, and LiveMathematicianBench $37.6 \rightarrow 66.9$ on GPT–5.5; SpreadsheetBench $9.3 \rightarrow 23.9$ ($\times 2.6$) on Qwen3.5–4B; and ALFWorld $34.3 \rightarrow 69.4$ ($\times 2.0$) on GPT–5.4-nano.

The improvement is not specific to frontier-scale targets. Averaged over the six benchmarks, SKILLOPT lifts GPT–5.4 by $+12.7$ points, GPT–5.4-mini by $+15.4$, GPT–5.4-nano by $+26.7$, GPT–5.2 by $+16.6$, Qwen3.5–4B by $+19.2$, and Qwen3.6–35B-A3B by $+9.1$, for an average improvement of approximately $+17.6$ points per model. Small and weak target models benefit the most in relative terms (e.g. GPT–5.4-nano nearly doubles on DocVQA and triples on ALFWorld), which is consistent with the view that a compact skill artifact can supply procedural knowledge that small models do not yet hold in weights.

The same optimization interface is also effective under tool-backed execution. On the Codex harness, SKILLOPT is best on all five evaluated benchmarks for GPT–5.5, with average gain $+24.8$ points over no skill and $+14.0$ over the next-best baseline (EvoSkill). On the Claude Code harness, it is best on all five benchmarks for GPT–5.5, with average gain $+19.1$ over no skill and $+3.2$ over EvoSkill, while EvoSkill itself already lifts the five-benchmark average from 57.8 to 73.7. The two ALFWorld cells under harness rows are left blank because ALFWorld requires persistent embodied-environment interaction that is not represented in the standard Codex / Claude Code adapters; we therefore report harness results on search, spreadsheets, document QA, multimodal QA, and math.

Taken together, the table supports a strong empirical claim: across direct chat and two tool-execution harnesses, across seven target models, and on procedural and factual benchmarks alike, optimizing a single compact skill artifact under bounded text-space training is the strongest no-weight-update adaptation strategy among the baselines we consider. The main gains come from feedback-driven skill editing rather than from a better one-shot prompt: human and LLM skills can help when prior instructions happen to match the benchmark, but they cannot correct failures after observing rollouts; Trace2Skill mines trajectory lessons without a held-out gate; TextGrad and GEPA optimize prompts but not a persistent skill artifact; and EvoSkill, the strongest harness-side competitor, lacks both bounded textual learning rates and rejected-edit memory. ProTeGi is likewise below SKILLOPT in all 12 of its cells and ends below its initial skill in three (Appendix G). These comparisons support the central design choice—keep the target model, harness, and evaluator fixed, and optimize only the reusable skill artifact.

4.2 Ablations

Table 4, Figure 3, and Table 2 test the design choices in the optimizer using GPT–5.5 as both the target and the optimizer. The overall message is that SKILLOPT benefits from sufficient evidence, a bounded textual learning rate, rejected-edit feedback, and epoch-wise slow/meta update. SearchQA has limited headroom and is therefore stable across many settings (most cells fluctuate inside a ± 1.5 point band), while SpreadsheetBench and LiveMathematicianBench expose the trade-off between learning useful procedures and over-editing the skill. The one-factor sweeps (Appendix F.2) show that procedural benchmarks keep improving with more training data, while results are flat across batch sizes, moderate edit budgets, and schedules.

Epoch-wise slow/meta update (panel f, Table 2, Figure 3). The slow/meta update supplies longer-horizon guidance beyond the current rollout batch. Slow-update sampling (panel f) places the default at 20 examples per epoch (87.1, 77.5, and 61.3), with 5, 10, and 40 each within ± 2.7 points. In the matched default component row, removing the rejected-edit buffer lowers scores by 1.6, 4.6, and 2.4 points on SearchQA, SpreadsheetBench, and LiveMath, respectively, supporting its role as a stabilizer for the default loop rather than as an extra deployment-time mechanism. The slow/meta ablation rows are sharper: removing both meta skill and slow update drops SpreadsheetBench from 77.5 to 55.0 (-22.5 points), the largest degradation in the ablation suite. Figure 3 (Appendix F.2) complements these numerical ablations: validation checkpoints track held-out test performance across epochs, confirming that the gate tends to select skills that generalize rather than skills that only fit the selection split.

4.3 Analysis and Transfer

Table 3 asks whether an optimized skill behaves like a reusable artifact rather than a task-specific prompt. We test three shifts: deploying a skill across model scales (Table 3a), moving it across execution harnesses (Table 3b), and applying it to nearby math benchmarks, including OlympiadBench [43] and Omni-MATH [14] (Table 3c).

Cross-model transfer. Table 3(a) reports four *genuine* cross-model transfers: SpreadsheetBench skills trained on GPT–5.4 transfer to GPT–5.4-mini ($+9.4$) and GPT–5.4-nano ($+3.0$), and LiveMath skills transfer to GPT–5.4-mini ($+4.5$) and GPT–5.4-nano ($+5.6$). All four cross-model rows are positive, and on one of them the transferred skill *surpasses* the in-domain SKILLOPT reference (LiveMath GPT–5.4-nano: 28.8 transferred vs. 27.2 in-domain), suggesting that some learned procedures are target-model agnostic. The remaining cross-model rows still recover a useful fraction of the in-domain gain—e.g. SpreadsheetBench GPT–5.4-mini retains 82% of the in-domain gain ($+9.4$ of $+11.4$)—and no row falls below the target's no-skill baseline.

Cross-harness transfer. The harness-shift rows in Table 3(b) are the clearest deployment signal. A SpreadsheetBench skill trained inside the Codex loop transfers to Claude Code with absolute gain $+59.7$ over the Claude Code no-skill baseline ($22.1 \rightarrow 81.8$, slightly exceeding the in-domain Claude Code SKILLOPT reference of 80.4), and the symmetric Claude-Code→Codex transfer adds $+43.6$ on top of the Codex baseline ($27.5 \rightarrow 71.1$). On LiveMath, the Codex→Claude Code transfer is smaller ($+1.6$ over a 40.8 baseline) but still positive, while the Claude-Code→Codex transfer adds $+12.8$ ($35.2 \rightarrow 48.0$). Because the two harnesses expose different tool/file APIs and command surfaces, these positive transfers suggest that the learned rules are not only harness-specific command recipes. In SpreadsheetBench especially, the transferred skill appears to encode workbook-level procedures such as structure-first inspection, formula-aware verification, and static-value materialization, so the cost of optimizing a skill in one execution environment can be amortized across related deployment environments.

Cross-benchmark transfer. Cross-benchmark transfer is the strictest of the three shifts: source and target benchmarks share only the broad task family (math). On the OlympiadBench→Omni-MATH direction reported in Table 3(c), the transferred skill is positive on all three model scales we evaluate, with gains of $+3.7$ on GPT–5.4, $+1.8$ on GPT–5.4-mini, and $+1.3$ on GPT–5.4-nano. These rows are smaller than the in-domain and cross-harness transfers—unsurprisingly, since they require the optimized skill to retain useful procedural knowledge after both the test instances and the answer-format conventions change—but they remain uniformly positive, supporting the intended interpretation that the optimized skill encodes reusable mathematical procedure rather than memorized benchmark-specific formatting. These shifts support portability within a deployment domain (the tested models, the Codex–Claude Code pair, and nearby benchmarks) rather than universal cross-domain transfer; Appendix K analyzes what transfers and how transfer fails, and we recommend re-validating transferred skills on target-side held-out data.

5 Conclusion

We presented SKILLOPT, a text-space optimizer that treats an external skill document as the trainable state for frozen LLM agents. By separating the target model that executes tasks from the optimizer that edits skills, and by using bounded edit budgets, minibatch reflection, held-out validation gates, rejected-edit buffers, and epoch-wise slow/meta update, SKILLOPT turns skill improvement into a controlled learning process rather than ad hoc prompt revision. Across six benchmarks, seven target models, and three execution modes, SKILLOPT is best or tied-best on 52 of 52 evaluated cells, lifts GPT–5.5 by $+23.5$ points on average over no skill in direct chat and by $+24.8/+19.1$ points under Codex and Claude Code harnesses, and beats the strongest per-cell baseline from human, LLM, Trace2Skill, TextGrad, GEPA, and EvoSkill skills by $+5.4$ points on average. Per-benchmark case studies show that these gains arise from compact ($< 2{,}000$ token), interpretable skill artifacts assembled from only 1–4 accepted edits, and that the deployed skills transfer across model scales, harnesses, and nearby benchmarks. These results suggest that compact natural-language skills can serve as a practical domain-adaptation layer for frontier agents, enabling reusable improvement without modifying model weights.

Acknowledgments and Disclosure of Funding

This work was supported by the National Natural Science Foundation of China (62506229), Natural Science Foundation of Shanghai under 25ZR1402268, and Shanghai QiYuan Innovation Foundation. Ziyang Gong, Weiquan Huang, Qihao Yang, Ziwei Zhou, Zisu Huang, and Yan Li did this work during internships at Microsoft Research Asia.

Competing interests. Yifan Yang, Xuemei Gao, Qi Dai, Bei Liu, Kai Qiu, Yuqing Yang, Dongdong Chen, and Chong Luo are employees of Microsoft.

References

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

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

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

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

[5] Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations*, 2023.

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

[7] 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.

[8] Yanna Jiang, Delong Li, Haiyu Deng, Baihe Ma, Xu Wang, Qin Wang, and Guangsheng Yu. Sok: Agentic skills–beyond tool use in llm agents. *arXiv preprint arXiv:2602.20867*, 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] 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.

[11] Xingyan Liu, Xiyue Luo, Linyu Li, Ganghong Huang, Jianfeng Liu, and Honglin Qiao. Skillforge: Forging domain-specific, self-evolving agent skills in cloud technical support. *arXiv preprint arXiv:2604.08618*, 2026.

[12] Shuaike Shen, Wenduo Cheng, Mingqian Ma, Alistair Turcan, Martin Jinye Zhang, and Jian Ma. Skillfoundry: Building self-evolving agent skill libraries from heterogeneous scientific resources. *arXiv preprint arXiv:2604.03964*, 2026.

[13] Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, et al. Gepa: Reflective prompt evolution can outperform reinforcement learning. *arXiv preprint arXiv:2507.19457*, 2025.

[14] Bofei Gao, Feifan Song, Zhe Yang, Zefan Cai, Yibo Miao, Qingxiu Dong, Lei Li, Chenghao Ma, Liang Chen, Runxin Xu, Zhengyang Tang, Benyou Wang, Daoguang Zan, Shanghaoran Quan, Ge Zhang, Lei Sha, Yichang Zhang, Xuancheng Ren, Tianyu Liu, and Baobao Chang. Omni-math: A universal olympiad level mathematic benchmark for large language models, 2024. URL https://arxiv.org/abs/2410.07985.

[15] Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. Automatic prompt optimization with "gradient descent" and beam search. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7957–7968, 2023.

[16] Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. Textgrad: Automatic "differentiation" via text. *arXiv preprint arXiv:2406.07496*, 2024.

[17] Weijia Song, Jiashu Yue, and Zhe Pang. Abstral: Automatic design of multi-agent systems through iterative refinement and topology optimization. *arXiv preprint arXiv:2603.22791*, 2026.

[18] Yufei He, Juncheng Liu, Yue Liu, Yibo Li, Tri Cao, Zhiyuan Hu, Xinxing Xu, and Bryan Hooi. Evotest: Evolutionary test-time learning for self-improving agentic systems. *arXiv preprint arXiv:2510.13220*, 2025.

[19] Shaokun Zhang, Jieyu Zhang, Jiale Liu, Linxin Song, Chi Wang, Ranjay Krishna, and Qingyun Wu. Offline training of language model agents with functions as learnable weights. *arXiv preprint arXiv:2402.11359*, 2024.

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

[21] Chenxi Wang, Zhuoyun Yu, Xin Xie, Wuguannan Yao, Runnan Fang, Shuofei Qiao, Kexin Cao, Guozhou Zheng, Xiang Qi, Peng Zhang, et al. Skillx: Automatically constructing skill knowledge bases for agents. *arXiv preprint arXiv:2604.04804*, 2026.

[22] 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.

[23] Hanrong Zhang, Shicheng Fan, Henry Peng Zou, Yankai Chen, Zhenting Wang, Jiayu Zhou, Chengze Li, Wei-Chieh Huang, Yifei Yao, Kening Zheng, et al. Evoskills: Self-evolving agent skills via co-evolutionary verification. *arXiv preprint arXiv:2604.01687*, 2026.

[24] Ziyu Ma, Shidong Yang, Yuxiang Ji, Xucong Wang, Yong Wang, Yiming Hu, Tongwen Huang, and Xiangxiang Chu. Skillclaw: Let skills evolve collectively with agentic evolver. *arXiv preprint arXiv:2604.08377*, 2026.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

[31] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

[32] Matthew Dunn, Levent Sagun, Mike Higgins, V Ugur Guney, Volkan Cirik, and Kyunghyun Cho. Searchqa: A new q&a dataset augmented with context from a search engine. *arXiv preprint arXiv:1704.05179*, 2017.

[33] 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.

[34] Krista Opsahl-Ong, Arnav Singhvi, Jasmine Collins, Ivan Zhou, Cindy Wang, Ashutosh Baheti, Owen Oertell, Jacob Portes, Sam Havens, Erich Elsen, et al. Officeqa pro: An enterprise benchmark for end-to-end grounded reasoning. *arXiv preprint arXiv:2603.08655*, 2026.

[35] Minesh Mathew, Dimosthenis Karatzas, and CV Jawahar. Docvqa: A dataset for vqa on document images. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, pages 2200–2209, 2021.

[36] Linyang He, Qiyao Yu, Hanze Dong, Baohao Liao, Xinxing Xu, Micah Goldblum, Jiang Bian, and Nima Mesgarani. Livemathematicianbench: A live benchmark for mathematician-level reasoning with proof sketches, 2026. URL https://arxiv.org/abs/2604.01754.

[37] 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.

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

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

[40] Qwen Team. Qwen3.6-35B-A3B: Agentic coding power, now open to all, April 2026. URL https://qwen.ai/blog?id=qwen3.6-35b-a3b.

[41] OpenAI. Codex: A cloud-based software engineering agent, 2025. URL https://openai.com/index/introducing-codex/. Accessed: 2026-05-06.

[42] Anthropic. Claude code: An ai coding agent system, 2025. URL https://www.anthropic.com/claude-code. Accessed: 2026-05-06.

[43] Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, et al. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. *arXiv preprint arXiv:2402.14008*, 2024.

[44] Yuning Wu, Jiahao Mei, Ming Yan, Chenliang Li, Shaopeng Lai, Yuran Ren, Zijia Wang, Ji Zhang, Mengyue Wu, Qin Jin, and Fei Huang. Writingbench: A comprehensive benchmark for generative writing. *arXiv preprint arXiv:2503.05244*, 2025.

[45] Jiaxin Ge, Zora Zhiruo Wang, Xuhui Zhou, Yi-Hao Peng, Sanjay Subramanian, Qinyue Tan, Maarten Sap, Alane Suhr, Daniel Fried, Graham Neubig, and Trevor Darrell. Autopresent: Designing structured visuals from scratch. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025.

[46] Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready AI agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*, 2025.

A Additional Method Details and Optimizer Prompts

This appendix gives the executable details behind SKILLOPT. The optimization loop keeps the task-execution model fixed and trains only a text skill document. A separate optimizer model reads rollout evidence, proposes patch-style edits, merges and ranks the edits, and submits each candidate skill to a held-out selection gate. The task-execution model only receives the current skill and the benchmark task; it does not see the optimizer prompts below.

B Limitations

SKILLOPT studies skill optimization as a lightweight alternative to model-weight adaptation, but it still has several practical limitations. First, the optimization loop relies on scored trajectories and a held-out selection split, so it is most directly applicable when the target task has automatic verifiers, exact-match metrics, executable checks, or otherwise reliable feedback signals. For open-ended domains where success is subjective, multi-dimensional, or costly to judge, the validation gate may require stronger human or model-based evaluation. Second, although the deployed artifact is only a compact `best_skill.md`, training the skill requires additional rollout computation and calls to an optimizer model; this cost is amortized when the same skill is reused, but may be less attractive for one-off tasks. Third, SKILLOPT intentionally optimizes a single portable skill rather than growing a large skill library or changing model weights. This design improves deployment simplicity, but a single skill may be insufficient for highly heterogeneous domains that require many disjoint procedures. Finally, optimized skills can encode domain-specific heuristics from the training distribution, so careful held-out evaluation remains necessary before transferring them to substantially different models, harnesses, or task settings.

C Outlook

SKILLOPT optimizes a single skill artifact for a single target domain; natural extensions include skill libraries that share infrastructure across domains, reuse of optimizer-side meta skills across benchmarks, reward-free or preference-driven validation gates for open-ended tasks, and self-distillation of optimized skills back into the target model as a stepping stone toward weight-level adaptation. We hope that treating the skill itself as the trainable object—rather than as a side artifact of prompting—will let future work apply the full toolkit of optimization (learning rates, schedules, regularization, curricula, validation) to a part of the agent stack that has so far been hand-engineered.

D Broader Impacts

SKILLOPT automates the procedural adaptation of frozen agents, including API-only models whose weights cannot be changed, and stores the result in a short natural-language skill that practitioners can read, audit, and edit, with no weight updates or additional model calls at deployment. The same automation can be misused: an optimizer driven by a harmful objective could encode harmful procedures into a skill for a capable tool-using agent, and because edits are accepted whenever they raise the measured score, flaws in the reward or evaluator can be exploited to produce a skill that games the metric rather than solving the task. Held-out validation limits overfitting to the training signal, although not flaws that the selection evaluator shares, and the deployed skill is short enough for a human to review before release. We therefore recommend reviewing every optimized skill and re-validating it on target-side held-out data before transferring it to a new model, harness, or task (Section 4.3).

E Experimental Protocol Details

Benchmarks and metrics. We use each benchmark's native evaluator and report hard success or exact-match accuracy on held-out test examples. SearchQA measures extractive question answering; SpreadsheetBench evaluates spreadsheet-oriented code and tool use; OfficeQA and DocVQA test local-document and multimodal-document reasoning; LiveMathematicianBench evaluates mathematical multiple-choice reasoning; and ALFWorld tests sequential decision making. Dataset-backed

benchmarks use deterministic train/selection/test splits, with a default 2:1:7 split when no benchmark-specific split is stated. The selection split is used only for model selection over candidate skills; all headline scores are computed on held-out test data.

Benchmark diversity. The benchmark suite is intentionally diverse—it spans single-round QA (SearchQA, DocVQA, LiveMathematicianBench MCQ), multi-turn tool loops with up to 24 tool calls (OfficeQA), multi-round codegen with up to 30 turns and a real `openpyxl`/`pandas` runtime (SpreadsheetBench, default `mode=multi`), and persistent embodied interaction with up to 50 steps per episode (ALFWorld). The reported numbers thus measure generalization, not validation-set fit.

Baselines. The no-skill baseline evaluates the frozen student without an optimized skill document. Human-skill and LLM-skill baselines use manually written and one-shot generated skills under the same evaluation protocol. Trace2Skill mines skill artifacts from training trajectories and evaluates the frozen student without SKILLOPT's iterative validation gate. TextGrad and GEPA are reflective prompt-optimization baselines for direct-chat settings. EvoSkill is included for the harness-backed comparison where a matched completed run is available. Entries not measured under the final aligned protocol are marked as –rather than mixed with incompatible runs.

Optimization protocol. Unless otherwise stated, SKILLOPT runs for four epochs with rollout batch size 40, reflection minibatch size 8, textual learning rate 4, cosine learning-rate decay with minimum rate 2, held-out validation gating, slow update enabled with 20 sampled examples, and optimizer-side meta skill enabled. The optimizer analyzes successes and failures separately, proposes patch-style skill edits, merges duplicate or contradictory proposals, ranks edits under the current learning-rate cap, and applies the selected edits to form a candidate skill. The candidate is evaluated on the selection split and is accepted only if it improves the current selection score; the best accepted skill is exported as `best_skill.md`. The student model, backend, harness, and benchmark evaluator remain fixed during optimization.

Default optimizer hyperparameters. Unless noted, SKILLOPT uses four epochs, rollout batch size 40 per step, reflection minibatch size 8 (with 16 analyst workers running reflections in parallel and a merge batch size of 8), textual learning rate $L_t = 4$ with cosine decay (floor $L_t = 2$, configurable schedules: constant, linear, cosine, autonomous), held-out validation gating (strictly greater than the current selection score—ties are rejected), slow update with 20 sampled tasks per epoch comparing previous-epoch and current-epoch skill, an optimizer-side meta skill that summarizes accepted/rejected patterns into teacher-only guidance, the `patch` edit mode (the alternative is `rewrite_from_suggestions`), and an optional rejected-edit buffer of recent failed proposals. Teacher reflection is allowed up to three refinement rounds per minibatch. Both teacher and student calls default to a `medium` reasoning effort. For benchmarks with tightly bounded training pools (LiveMathematicianBench: 35 training items per epoch with rollout batch 200; ALFWorld: 39 training tasks with 18 selection and 134 test environments), per-benchmark configs scale the batch sizes accordingly while keeping the same gate, scheduler, and slow/meta-update machinery.

Harnesses. Direct chat invokes the target model through a single chat completion call with the skill prepended to the system prompt. The Codex harness drives the target through the `codex` CLI in a workspace-write sandbox [41]; SKILLOPT renders the current skill to a per-task `SKILL.md` alongside task files and reads back a compact execution trace (`codex_trace_summary.txt`) that is included in the teacher reflection context, so the optimizer learns from *what the agent actually did*, not just its final answer. The Claude Code harness mirrors the same workspace contract through the `claude` CLI [42]. All three modes consume the same `best_skill.md` file format, which is what enables the cross-harness transfer experiments in Section 4.3.

Ablation protocol. One-factor ablations vary a single scalar or component while holding the remaining optimizer configuration fixed. The train-size ablation fixes the train/selection/test split to 2:1:7 and varies how much of the training partition is exposed to the optimizer. The 100% row uses the full training partition under the same split, so it is directly comparable to the smaller-subset rows. Component ablations remove or alter one mechanism at a time, including the edit budget, rejected-edit buffer, and epoch-wise slow/meta update.

Algorithm 1 SKILLOPT skill optimization

Require: Frozen training model M, optimizer model O, harness h, splits $D_{\text{train}}, D_{\text{sel}}, D_{\text{test}}$, initial skill s_0, epochs E, edit-budget schedule L_t, rollout batch size B, accumulation factor A, reflection minibatch size B_m

Ensure: Best validation-gated skill s_{best} and held-out test score

$s_{\text{cur}} \leftarrow s_0$, $s_{\text{best}} \leftarrow s_0$, $\mathcal{C} \leftarrow \emptyset$, $\mathcal{B} \leftarrow [\,]$, $m_{\text{meta}} \leftarrow \emptyset$
$\text{score}_{\text{cur}} \leftarrow \text{EVALUATE}(M, h, s_0, D_{\text{sel}})$; $\text{score}_{\text{best}} \leftarrow \text{score}_{\text{cur}}$
$\mathcal{C}[\text{HASH}(s_0)] \leftarrow \text{score}_{\text{cur}}$
for $e = 1$ to E **do**
    Shuffle D_{train} into rollout batches; reset $\mathcal{B} \leftarrow [\,]$
    for each optimization step in epoch e **do**
        Collect A rollout batches by executing $h(M, x, s_{\text{cur}})$ for sampled tasks x
        Split rollout evidence into failures and successes, then into minibatches of size B_m
        Ask O to analyze failure minibatches and produce failure patch proposals
        Ask O to analyze success minibatches and produce success patch proposals
        Ask O to merge failure proposals, merge success proposals, and perform a final failure-prioritized merge
        Ask O to rank merged edits and keep at most L_t edits
        Apply the selected edits to obtain a candidate skill $\tilde{s}$
        if $\text{HASH}(\tilde{s}) \in \mathcal{C}$ **then**
            $\text{score}_{\text{cand}} \leftarrow \mathcal{C}[\text{HASH}(\tilde{s})]$
        else
            $\text{score}_{\text{cand}} \leftarrow \text{EVALUATE}(M, h, \tilde{s}, D_{\text{sel}})$
            $\mathcal{C}[\text{HASH}(\tilde{s})] \leftarrow \text{score}_{\text{cand}}$
        end if
        if $\text{score}_{\text{cand}} > \text{score}_{\text{cur}}$ **then**
            $s_{\text{cur}} \leftarrow \tilde{s}$; $\text{score}_{\text{cur}} \leftarrow \text{score}_{\text{cand}}$
            if $\text{score}_{\text{cand}} > \text{score}_{\text{best}}$ **then**
                $s_{\text{best}} \leftarrow \tilde{s}$; $\text{score}_{\text{best}} \leftarrow \text{score}_{\text{cand}}$
            end if
        else
            Add rejected edits and observed failure patterns to $\mathcal{B}$
        end if
    end for
    if $e \geq 2$ and slow update is enabled **then**
        Compare the same sampled tasks under the previous and current epoch-end skills
        Ask O for protected longitudinal guidance; validate the injected guidance through D_{sel}
    end if
    if $e \geq 2$ and optimizer memory is enabled **then**
        Ask O to update m_{meta} for future edit generation and selection
    end if
end for
$\text{score}_{\text{test}} \leftarrow \text{EVALUATE}(M, h, s_{\text{best}}, D_{\text{test}})$
return s_{best}, $\text{score}_{\text{test}}$

E.1 Optimization Procedure

Algorithm 1 expands the procedure used in the experiments. The central state variables are the current skill s_{cur}, the best validation-gated skill s_{best}, a selection-score cache $\mathcal{C}$, a step buffer $\mathcal{B}$ containing rejected edits and observed failure patterns, and an optimizer-side meta skill m_{meta} used only to guide future edit generation.

E.2 Optimizer Prompt Contracts

The following blocks reproduce the operational prompt contracts used by the optimizer model, with terminology normalized to the paper's optimizer/training-model framing. The prompts require JSON outputs so that edits can be parsed, filtered, applied, and validated without manual intervention.

E.2.1 Failure analysis: `analyst_error.md`

```
You are an expert failure-analysis agent for AI agent tasks.

You will be given MULTIPLE failed agent trajectories from a single minibatch
and the current skill document.
Your job is to identify the most important COMMON failure patterns across
the batch and propose a concise set of skill edits.

## Analysis Process
1. Read ALL trajectories in the minibatch.
2. Identify the most prevalent, systematic failure patterns across them.
3. For each pattern, classify its failure type.
4. Propose skill edits that address the COMMON patterns, not individual edge cases.
5. Edits must be generalizable; do not hardcode task-specific values.
6. Only patch gaps in the skill; do not duplicate existing content.

You will be told the maximum number of edits (the budget L). Produce AT MOST L edits,
focusing on the highest-impact patterns. You may produce fewer if warranted.

Respond ONLY with a valid JSON object (no markdown fences, no extra text):
{
  "batch_size": <number of trajectories analysed>,
  "failure_summary": [
    {"failure_type": "<type>", "count": <int>, "description": "<one-line>"}
  ],
  "patch": {
    "reasoning": "<why these edits address the batch's common failures>",
    "edits": [
      {"op": "append",       "content": "<markdown to add at end of skill>"},
      {"op": "insert_after", "target": "<exact heading/text to insert after>",
       "content": "<markdown>"},
      {"op": "replace",      "target": "<exact text to replace>",
       "content": "<replacement>"},
      {"op": "delete",       "target": "<exact text to remove>"}
    ]
  }
}
Only include edits that are needed. "edits" can be an empty list if no patch is warranted.

IMPORTANT: The skill document may contain a section between
<!-- SLOW_UPDATE_START --> and <!-- SLOW_UPDATE_END --> markers.
This is a PROTECTED section managed by a separate slow-update process.
Do NOT propose any edits that target, modify, or delete content within these markers.
```

E.2.2 Success analysis: `analyst_success.md`

```
You are an expert success-pattern analyst for AI agents.

You will be given MULTIPLE successful agent trajectories from a single minibatch
and the current skill document. Your job is to identify generalizable behavior
patterns that are COMMON across the batch and worth encoding in the skill.

## Rules
- Only propose patches for patterns NOT already covered in the skill.
- Focus on patterns that appear across MULTIPLE trajectories in the batch.
- Be concise. Patterns must generalize beyond specific tasks.
- Prefer reinforcing existing sections over adding new top-level sections.

You will be told the maximum number of edits (the budget L). Produce AT MOST L edits,
focusing on the most broadly applicable patterns. You may produce fewer if warranted.

Respond ONLY with a valid JSON object:
{
  "batch_size": <number of trajectories analysed>,
  "success_patterns": ["<pattern 1>", "<pattern 2>"],
  "patch": {
    "reasoning": "<why these patterns are worth encoding>",
```

```
    "edits": [
      {"op": "append",       "content": "<markdown>"},
      {"op": "insert_after", "target": "<heading/text>", "content": "<markdown>"},
      {"op": "replace",      "target": "<old text>",     "content": "<new text>"},
      {"op": "delete",       "target": "<exact text to remove>"}
    ]
  }
}
"edits" may be empty if the skill already covers all observed patterns.

IMPORTANT: The skill document may contain a section between
<!-- SLOW_UPDATE_START --> and <!-- SLOW_UPDATE_END --> markers.
This is a PROTECTED section managed by a separate slow-update process.
Do NOT propose any edits that target, modify, or delete content within these markers.
```

E.2.3 Failure merge: `merge_failure.md`

```
You are a skill-edit coordinator. You receive multiple independently-proposed patches
from FAILURE analysis of agent trajectories. Merge them into ONE coherent,
non-redundant patch.

Merge guidelines:
1. Deduplicate: keep the best-worded version of similar edits.
2. Resolve conflicts: if patches contradict on the same point,
   choose the one with stronger justification or synthesize both.
3. Preserve unique insights: include all non-redundant corrective edits.
4. Prevalent-pattern bias: edits appearing consistently across multiple patches
   address systematic failures; preserve them with HIGH priority.
   Edits from only one patch may be discarded if task-specific.
5. Independence: no two edits in the merged patch may target the same text region.
6. Support count: for each merged edit, estimate how many source patches support it.
7. PROTECTED SECTION: The skill may contain a section between
   <!-- SLOW_UPDATE_START --> and <!-- SLOW_UPDATE_END --> markers.
   Do NOT merge or produce any edits that target content within these markers.

Respond ONLY with a valid JSON object:
{
  "reasoning": "<summary of key consolidation decisions>",
  "edits": [
    {
      "op": "append|insert_after|replace|delete",
      "target": "<if insert_after or replace or delete>",
      "content": "<markdown>",
      "support_count": <integer>,
      "source_type": "failure"
    }
  ]
}
```

E.2.4 Success merge: `merge_success.md`

```
You are a skill-edit coordinator. You receive multiple independently-proposed patches
from SUCCESS analysis of agent trajectories. Merge them into ONE coherent patch
that reinforces effective patterns.

Merge guidelines:
1. Deduplicate: keep only the most generalizable version of similar patterns.
2. Be conservative: success-driven patches reinforce existing behavior.
   Only include edits for patterns NOT already in the skill.
3. Prevalent-pattern bias: patterns seen across many successful trajectories
   are most worth encoding.
4. Support count: estimate how many source patches support each merged edit.
5. PROTECTED SECTION: The skill may contain a section between
   <!-- SLOW_UPDATE_START --> and <!-- SLOW_UPDATE_END --> markers.
   Do NOT merge or produce any edits that target content within these markers.

Respond ONLY with a valid JSON object:
{
```

```
  "reasoning": "<summary>",
  "edits": [
    {
      "op": "append|insert_after|replace|delete",
      "target": "<if needed>",
      "content": "<markdown>",
      "support_count": <integer>,
      "source_type": "success"
    }
  ]
}
```

E.2.5 Final merge: `merge_final.md`

```
You are a skill-edit coordinator performing the FINAL merge. You receive two
pre-merged patch groups:
1. Failure-driven patches (corrective, high priority)
2. Success-driven patches (reinforcement, lower priority)

Merge guidelines:
1. FAILURE PATCHES TAKE PRIORITY: the primary goal of skill reflection is to
   fix failures. Failure-driven edits should be preserved unless they directly
   conflict with a well-supported success pattern.
2. Deduplicate: if a failure edit and success edit cover the same point,
   keep the failure version.
3. Preserve success insights: include success edits that cover patterns
   NOT addressed by failure edits.
4. Higher-level merges represent broader consensus: edits that survived
   previous merge rounds should be given priority.
5. Carry forward support_count and source_type for each edit.
6. PROTECTED SECTION: The skill may contain a section between
   <!-- SLOW_UPDATE_START --> and <!-- SLOW_UPDATE_END --> markers.
   Do NOT merge or produce any edits that target content within these markers.

Respond ONLY with a valid JSON object:
{
  "reasoning": "<summary of priority decisions>",
  "edits": [
    {
      "op": "append|insert_after|replace|delete",
      "target": "<if needed>",
      "content": "<markdown>",
      "support_count": <integer>,
      "source_type": "failure|success"
    }
  ]
}
```

E.2.6 Ranking and selection: `ranking.md`

```
You are an expert edit-ranking optimizer for a skill optimization system. You receive
a skill document and a pool of proposed edits. Your job is to RANK the edits by
importance and select the top ones.

Ranking criteria (in order of priority):
1. Systematic impact: edits that address widespread, recurring failure patterns
   across many tasks should rank highest. A rule that fixes 50% of failures beats
   one that fixes a single edge case.
2. Complementarity: edits that fill gaps in the current skill, not duplicate
   existing content, rank higher.
3. Generality: edits phrased as general principles rank higher than those
   tied to specific question types or entities.
4. Actionability: edits with clear, concrete guidance rank higher than vague advice.

You will be told how many edits to select (the budget).

Respond ONLY with a valid JSON object:
{
```

```
  "reasoning": "<brief justification for your ranking decisions>",
  "selected_indices": [<0-based indices of the top edits, in priority order>]
}
```

E.2.7 Slow update: `slow_update.md`

```
You are a strategic skill advisor for an AI agent optimization system.

Your role is different from the per-step analyst. The per-step analyst sees
individual trajectories and proposes local patches. YOU see how the skill has
evolved across an entire epoch by comparing the SAME tasks under two consecutive
skill versions. This longitudinal view lets you identify systemic drift,
regressions, and persistent blind spots that step-level edits cannot catch.

## What You Receive

1. Previous epoch's skill and current epoch's skill, to see what changed.
2. Longitudinal comparison: the same 20 training tasks rolled out under both skills,
   categorized into regressions, persistent failures, improvements, and stable successes.
3. Previous slow update guidance, if any: the guidance written at the end of the
   last epoch.

## Your Process

1. Reflect on the previous guidance, if provided:
   - Which parts of the previous guidance were effective?
   - Which parts failed or backfired?
   - Were there blind spots the previous guidance missed entirely?

2. Write updated guidance that:
   - Retains and strengthens parts of the previous guidance that proved effective.
   - Revises or removes parts that were ineffective or counterproductive.
   - Adds new instructions to address newly observed regressions and persistent failures.

## Output Requirements

Write a strategic guidance block that will OVERWRITE the previous guidance
in the protected section of the skill document. This section is READ-ONLY to
all subsequent step-level optimization; only this epoch-boundary process can
overwrite it at the next epoch boundary.

Your guidance must:
- Be written as direct, actionable instructions to the training model.
- Prioritize: (1) preventing regressions, (2) fixing persistent failures,
  (3) reinforcing successful patterns.
- NOT duplicate content already in the main skill body; complement it.
- Address the training model directly, for example: "When you encounter X, always do Y."

Respond ONLY with a valid JSON object:
{
  "reasoning": "<reflection on previous guidance AND analysis of longitudinal comparison>",
  "slow_update_content": "<the exact guidance text to insert into the protected section>"
}
```

E.2.8 Optimizer memory: `meta_skill.md`

```
You are an optimizer coach for an AI agent skill optimization system.

Your job is not to solve tasks directly and not to write training-model-facing
skill rules. Your job is to write a compact optimizer-side meta skill that helps
future optimizer calls produce better skill edits in this environment.

## What You Receive

1. The previous epoch's last-step skill.
2. The current epoch's last-step skill.
3. A longitudinal comparison on the SAME sampled tasks under those two skills.
4. The previous optimizer memory, if one existed.
```

```
## Your Goal

Write a concise optimizer memory that improves future optimizer behavior in stages
such as failure analysis, success analysis, patch merging, and edit ranking.

This optimizer memory should capture things like:
- Which kinds of edits tend to help in this environment.
- Which kinds of edits tend to be too vague, redundant, brittle, or harmful.
- What level of abstraction works best for rules here.
- What failure-repair patterns should be prioritized.
- What regression risks future optimizer calls should guard against.

## Important Constraints

- Address the FUTURE OPTIMIZER directly, not the training model.
- Focus on how to write better edits and organize better skill updates.
- Use evidence from the adjacent-epoch comparison, not generic advice.
- Keep it compact and high-signal. Prefer a few durable principles.
- Revise or remove parts of the previous optimizer memory if they did not help.
- Do not output training-model-facing task instructions.

Respond ONLY with a valid JSON object:
{
  "reasoning": "<brief reflection on what editing directions helped or hurt>",
  "meta_skill_content": "<compact optimizer guidance for future edits>"
}
```

E.3 Patch Representation and Safeguards

Patch-mode optimization restricts each update to four atomic operations: `append`, `insert_after`, `replace`, and `delete`. Each merged edit also records a support count and a source type, allowing ranking to prefer edits that survive independent analyses and hierarchical merges. The edit budget L_t acts as a textual learning rate: it limits how many proposed edits can be applied at a step, preserving continuity between adjacent skills.

The protected slow-update section, delimited by `SLOW_UPDATE_START` and `SLOW_UPDATE_END`, is off limits to all step-level prompts. Only the epoch-boundary slow-update process may rewrite that section, and the rewritten skill still passes through the same held-out selection gate before it can become the current skill. Rejected candidates are not discarded entirely: their failure patterns and rejected edits are stored in the step buffer so that later optimizer calls can avoid repeating harmful changes.

E.4 Design Principles

The implementation follows five design principles. First, the task-execution model is fixed; only the text skill changes. Second, every candidate skill is evaluated on a selection split before acceptance, which prevents unvalidated reflection from accumulating. Third, minibatch analyses are merged hierarchically so that the final edits represent recurring evidence rather than single examples. Fourth, the edit budget serves as a learning-rate analogue, allowing larger early changes and smaller late refinements. Fifth, the deployed skill remains lightweight and inspectable, while the optimizer-side meta skill stays separate from the skill shown to the task-execution model.

F Additional Results and Analyses

This appendix collects analyses from Section 4 that do not fit in the main text.

F.1 Alternative Explanations of the Main Results

The per-cell baselines clarify what drives the gains. The effect is not simply prompt length: human skills are already 145–516 tokens long and often exceed the one-shot LLM skill, yet they are beaten in every direct-chat model row while the learned artifacts remain compact (Table 6). It is also not

(a) Training set size

Setting	SearchQA	Spreadsheet	LiveMath
1 example	81.0	47.5	59.1
20% train	84.1	69.0	65.9
40% train	**86.1**	73.5	64.8
80% train	**86.1**	77.6	67.0
100% train	84.1	**78.0**	**70.5**

(b) Mini-batchsize

Setting	SearchQA	Spreadsheet	LiveMath
1	85.9	75.4	60.5
2	86.3	77.1	54.8
4	86.9	75.4	**64.5**
8	**87.1**	77.5	61.3
16	87.0	**77.9**	61.3
32	86.9	77.5	58.9

(c) Batchsize

Setting	SearchQA	Spreadsheet	LiveMath
8	85.1	76.8	58.1
24	86.4	77.1	**62.9**
40	87.1	**77.5**	61.3
56	86.5	76.8	56.5
full epoch	**87.2**	75.0	53.2

(d) Learning rate

Setting	SearchQA	Spreadsheet	LiveMath
lr=1	85.5	77.5	62.1
lr=2	86.7	77.5	60.5
lr=4	86.5	**78.2**	56.5
lr=8	**87.0**	73.6	**66.9**
lr=16	86.8	**78.2**	65.3

(e) Learning-rate scheduler

Setting	SearchQA	Spreadsheet	LiveMath
constant	**87.3**	**80.7**	62.1
cosine	87.1	77.5	61.3
linear	87.2	72.9	**62.9**

(f) Slow-update samples

Setting	SearchQA	Spreadsheet	LiveMath
5	86.8	76.4	64.5
10	86.4	74.3	**65.3**
20	**87.1**	**77.5**	61.3
40	86.9	75.4	54.8

Table 4: Hyperparameter analysis for the text optimizer. Each panel changes one scalar or scheduling factor from the default setting unless noted. Panel (a) fixes the split to $4{:}1{:}5$ train/selection/test; the 1-example, 20%, 40%, and 80% rows use subsets of the training partition, and the 100% row reuses the completed $4{:}1{:}5$ split-ratio run. Panel (b) sweeps the reflection mini-batchsize B_m; panel (c) sweeps the rollout batchsize B.

only optimizer capacity: SKILLOPT leads every baseline even for GPT–5.4-nano, and the optimizer-strength analysis in Table 5 shows that a target-matched optimizer recovers much of the gain. Finally, the harness results show that the method is not just exploiting one skill format: EvoSkill already improves the Codex SpreadsheetBench cell from 27.5 to 67.5, but SKILLOPT adds another $+17.5$ points ($67.5 \rightarrow 85.0$). The gains are largest on procedural benchmarks, where reusable rules about tool use and output formatting matter most, but they also appear on factual and multimodal benchmarks.

F.2 Hyperparameter Sweeps

Evidence and batch sizes (panels a, b, c). Panel (a) shows that procedural benchmarks reward more training evidence: SpreadsheetBench climbs $47.5 \rightarrow 78.0$ and LiveMathematicianBench climbs $59.1 \rightarrow 70.5$ as the optimizer sees $1 \rightarrow 100\%$ of the training partition, while SearchQA saturates at roughly $84-86$ after 20% already. Panel (b) shows the same robustness in the other direction: varying the reflection mini-batchsize from 1 to 32 keeps SearchQA inside $85.9-87.1$ and SpreadsheetBench inside $75.4-77.9$, with the default $B_m{=}8$ at or near the top on all three benchmarks. Panel (c) is equally flat in the rollout-batchsize dimension—moving from $B{=}8$ to a full epoch keeps SearchQA inside $85.1-87.2$ and SpreadsheetBench inside $75.0-77.5$. Together this means the headline gains are not the product of a fragile prompt-search batch size, but a genuine effect of having enough scored evidence per update.

Textual learning rate and schedule (panels d, e). Panels (d) and (e) directly compare bounded textual learning to looser settings. Sweeping $L_t \in \{1, 2, 4, 8, 16\}$ shows that small or moderate edit budgets are competitive throughout: $L_t{=}4$ achieves $86.5/78.2/56.5$, the highest LiveMath score belongs to $L_t{=}8$ at 66.9, and the lowest score across all five settings is still only 85.5 on SearchQA. Panel (e) confirms this on the schedule axis: constant decay scores $87.3/80.7/62.1$, cosine $87.1/77.5/61.3$, and linear $87.2/72.9/62.9$, so the bounded-update story does not depend on a single specific scheduler. The qualitative claim is what matters: any moderate, bounded edit budget already beats baselines that rewrite the skill without a budget (Table 2, "without lr" row, $84.6/75.7/57.3$).

F.3 Gate Strictness and Edit Observability

The validation gate is intentionally strict: a candidate skill is accepted only when its selection-split score is *strictly greater than* the current selection score, so ties are rejected and the deployed skill never silently drifts. This conservative criterion makes rejected edits informative negative feedback rather than hidden state. Operationally, every step also records an `edit_apply_report.json` containing per-edit accept/skip status, so the source of every change to `best_skill.md` is recoverable after the fact. The epoch-wise slow/meta update writes into a markup-fenced protected region of the skill document that step-level edits cannot overwrite, separating the fast intra-epoch update from the slower cross-epoch consolidation; the optimizer-side meta skill lives only in the teacher's reflection context and is never shipped with the deployed artifact. These implementation choices explain

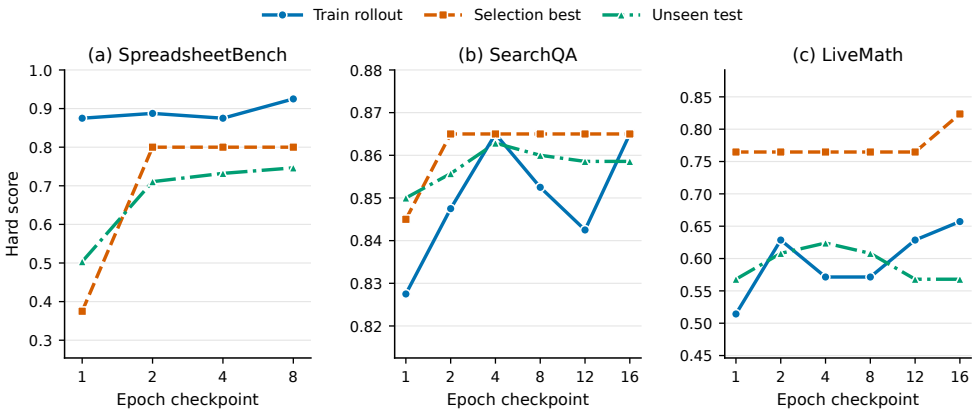

Figure 3: Performance trends across epoch checkpoints on three benchmarks: (a) SpreadsheetBench, (b) SearchQA, and (c) LiveMath. For each checkpoint, we report the training rollout score, the selection-best score on the validation set, and the final performance on the unseen test set. The results show how skill quality evolves during optimization and whether the checkpoint preferred by validation selection aligns with the checkpoint that yields the best generalization to the test set.

why removing both meta skill and slow update is especially damaging on SpreadsheetBench: it removes the long-horizon evidence stream and the protected-region contract that keeps local edits from overwriting durable procedural lessons.

F.4 Effect of Optimizer Strength

Because the optimizer in SKILLOPT runs only during the offline training loop and is never invoked at deployment, optimizer choice is a training-time lever: a stronger optimizer can improve the deployed skill without raising the inference cost of using that skill. The deployed artifact is still a static `best_skill.md` that calls only the target model. Table 5 quantifies this lever by running the same loop with two optimizer regimes—a strong frontier optimizer (GPT–5.5) and a target-matched optimizer that shares the target model—while holding the rollout batches, validation gate, bounded edit budget, rejected-edit buffer, and slow/meta update identical.

Two observations follow. *First*, the stronger optimizer produces larger absolute gains on every (benchmark, target) cell we test: GPT–5.4-nano lifts by $+19.0$ vs. $+11.9$ on SpreadsheetBench and $+19.0$ vs. $+14.1$ on SearchQA, and GPT–5.4-mini follows the same ordering ($+11.4$ vs. $+7.1$ on SpreadsheetBench, $+4.3$ vs. $+2.4$ on SearchQA). The bounded-edit, validation-gated loop is what makes this monotone: without the gate, a stronger optimizer could just as easily push larger but harmful rewrites. *Second*, the target-matched optimizer is far from collapsed—it recovers 56–74% of the strong-optimizer gain across the four cells, confirming that SKILLOPT is not a distillation pipeline from a stronger teacher into a weaker student: the optimization loop itself contributes substantial value on top of whatever the optimizer can already do. The practical implication is that a high-capacity frontier optimizer is the right default whenever it is available—it costs only training-time API calls and adds nothing to deployment—while the same loop remains effective if the budget forces a target-matched optimizer instead.

F.5 Learned Skills: Compactness, Cost, and Examples

A central premise of SKILLOPT is that the trainable object should remain a small, inspectable text document. Tables 1–5 demonstrate that the optimizer is effective; this subsection asks what its output actually looks like and what it costs. We characterize the learned artifact on three axes—compactness, edit economy, and cost-per-point—and then show one representative learned rule per benchmark to illustrate what kind of procedural knowledge survives the bounded-update loop.

Compactness. The final skills are uniformly small. Across the six benchmarks in Table 6, the final `best_skill.md` ranges from 379 tokens (LiveMathematicianBench) to 1,995 tokens (SpreadsheetBench), with a median of roughly 920 tokens. Even the longest learned skill is well below a typical

Benchmark	Target	Baseline	Strong optimizer (GPT–5.5)	Target-matched optimizer
SpreadsheetBench	GPT–5.4-mini	36.1	**47.5**$_{+11.4}$	43.2$_{+7.1}$
	GPT–5.4-nano	23.5	**42.5**$_{+19.0}$	35.4$_{+11.9}$
SearchQA	GPT–5.4-mini	75.9	**80.2**$_{+4.3}$	78.3$_{+2.4}$
	GPT–5.4-nano	55.8	**74.8**$_{+19.0}$	69.9$_{+14.1}$

Table 5: Effect of optimizer strength. Each (benchmark, target) pair is optimized either by a strong frontier optimizer (GPT–5.5, bolded) or by a target-matched optimizer that shares the target model; everything else in the SKILLOPT loop is held fixed. Gains over the target's no-skill baseline are shown as small green subscripts; the same baseline is used for both optimizer settings within a row. The optimizer runs only during offline training, so the stronger-optimizer column adds zero cost at deployment.

Benchmark	Initial (tok)	Final (tok)	Edits	Train tokens	Cost / pt
SearchQA	16	857	4	213.8M	37.9M
SpreadsheetBench	224	1,995	4	21.4M	0.6M
OfficeQA	145	883	1	20.8M	1.1M
DocVQA	81	959	3	188.2M	46.4M
LiveMath	154	379	1	23.2M	3.6M
ALFWorld	516	1,321	2	59.3M	15.9M

Table 6: Cost and edit economy of the GPT–5.5 / GPT–5.5 (student / teacher) skill runs. Initial and final `best_skill.md` lengths are in tokens; **Edits** is the number of accepted bounded updates; **Cost / pt** is training tokens per absolute test-point gain. One representative learned rule per benchmark is shown in Figure 4.

system-prompt budget for modern frontier models, and the shortest one fits inside a single screen. The growth from initial to final skill is moderate ($\times 2.5$ to $\times 53$ depending on whether the initial skill was a one-liner or a paragraph), but the final size in absolute tokens stays small enough that a domain practitioner can read, audit, and edit the deployed artifact in minutes.

Edit economy. A second striking pattern is that the gains come from very few accepted edits. Across the six benchmarks, the number of edits actually committed to `best_skill.md` during optimization is between 1 and 4 (median 2.5). LiveMathematicianBench's $+29.3$ point gain over no skill arises from a *single* accepted edit, and OfficeQA's $+39.0$ point gain similarly arises from one accepted edit. This is direct evidence that the validation gate is doing real work: the optimizer model proposes many more edits per epoch, but only a handful pass the held-out check and survive into the deployed skill. The bulk of the optimizer's text-space search is thus rejected, captured by the rejected-edit buffer (Section 3.5) for future use, and never reaches the target model. The deployed skill is correspondingly compact rather than the union of every reflection.

Cost per point of test-set gain. The training-token column quantifies the cost of operating the loop. Two regimes are visible. Procedural benchmarks where rollouts are short and cheap—SpreadsheetBench, OfficeQA, LiveMathematicianBench—reach 0.6–3.6M training tokens per absolute test-set point, even though the absolute gains on these benchmarks are the largest (e.g. $+39.0$ points on OfficeQA at 1.1M tokens / point, total 20.8M tokens). Benchmarks with longer trajectories or richer multimodal context—SearchQA (37.9M / pt) and DocVQA (46.4M / pt)—cost an order of magnitude more per point. The important deployment distinction is that this cost is paid once during skill training; after export, the optimized `best_skill.md` adds no optimizer calls, no weight updates, and only a compact text artifact to the target agent.

What does a learned skill actually say? Figure 4 reproduces one representative learned rule per benchmark, taken verbatim from the final `best_skill.md` of each case study in Table 6. Three observations stand out. First, the rules are *procedural* rather than *instance-specific*: none of them name a specific question, file, or entity. Second, they consistently encode the discipline that frontier models lack zero-shot: answer-format constraints (OfficeQA, LiveMathematicianBench), evidence binding to a specific visual region (DocVQA), workbook-structure-first reasoning (SpreadsheetBench), search-frontier discipline (ALFWorld), and canonical-entity choice (SearchQA). Third, they read like rules a

SearchQA. "Infer the expected answer type from clue wording, then choose the shortest canonical entity supported by co-occurring distinctive evidence."

SpreadsheetBench. "Inspect workbook structure and formulas, then write evaluated static values across the full requested target range instead of relying on Excel recalculation."

OfficeQA. "Treat oracle parsed pages as primary evidence, lock table/date/unit context, and output exactly the requested rounded value without extra labels."

DocVQA. "For tables, forms, charts, and legends, first bind the question to the exact visual row/header/field, then copy only the aligned answer span."

LiveMathematicianBench. "In strongest-statement MCQs, rank choices by theorem strength and prefer a justified stronger-result option over true but weaker corollaries."

ALFWorld. "Keep a horizon-aware visited/frontier ledger, diversify search after repeated same-type failures, and avoid revisiting the destination until holding the target."

Figure 4: Representative learned rules, one per benchmark, extracted from the final `best_skill.md` of the GPT–5.5 / GPT–5.5 runs in Table 6. Each rule is verbatim from the deployed skill. Notably, every rule is procedural rather than instance-specific, and several encode forms of discipline (answer formatting, evidence binding, search-frontier management) that frontier models do not apply zero-shot.

thoughtful human practitioner would write after a day with the benchmark—except they are produced automatically by the optimizer and validated edit-by-edit on held-out data.

Implications. Together, the four observations above support a stronger version of the central claim. Compactness ($< 2{,}000$ tokens) and edit economy (1–4 accepted edits) mean the deployed artifact is interpretable. Cost-per-point (0.6M–46.4M tokens / point) shows that the training cost is measurable and paid before deployment. The shape of the learned rules—procedural, generalizable, and consistent with what a thoughtful human practitioner would write—is evidence that text-space optimization with bounded updates and validation gating discovers transferable procedural knowledge rather than merely overfitting to the training split. This complements the cross-model, cross-harness, and cross-benchmark transfer evidence in Section 4.3: the artifact transfers because many of the rules it encodes are intrinsically transferable.

F.6 Qualitative Skill Evolution

We inspect two representative runs to understand what the optimized skill actually learns. The ALFWorld case uses GPT–5.4-nano as the student and GPT–5.5 as the teacher, while the Spreadsheet-Bench case uses GPT–5.5 as both the frozen student and optimizer model. In both cases, SKILLOPT does not replace the initial skill with an unrelated prompt. Instead, accepted edits add compact procedural constraints around recurring failure modes observed in rollout trajectories.

ALFWorld. The initial ALFWorld skill gives a generic household plan: search for the target object, pick it up, transform it if needed, and place it at the destination. The accepted edits make this plan more stateful and less loop-prone. The optimized skill learns exact object-name matching, so related objects such as mugs, cups, pans, and pots are not substituted for one another. It adds visited-location memory, so unvisited receptacles and surfaces are preferred over repeatedly checking likely but exhausted locations. It also adds destination memory, pick-two progress locks, and direct completion rules: once the agent can clean, heat, cool, place, or otherwise complete the next subgoal, it should take that admissible action instead of examining, closing, or verifying again. Qualitatively, the skill evolves from a general search-transform-place strategy into a finite-state execution policy with object identity, search memory, progress locks, and loop breakers. In this representative run, the selected skill improves ALFWorld held-out test performance from 49.3 to 74.6.

SpreadsheetBench. The initial SpreadsheetBench skill already instructs the agent to use Python spreadsheet libraries and preserve unrelated workbook content. The accepted edits turn this generic automation workflow into a workbook-forensics policy. The optimized skill learns to inspect the actual workbook rather than rely on previews, locate headers and target ranges across multiple sheets, normalize keys and cell types before lookup or aggregation, and preserve formatting during structural edits. It also adds a key rule for formula-style prompts: when the grader reads cell values, the

agent should compute and write evaluated static values, even if the prompt mentions formulas such as `INDEX/MATCH` or `XLOOKUP`. Later edits further require filling complete target ranges, including currently blank result cells, keeping helper computations in Python rather than adding workbook artifacts, and reopening the saved workbook to check boundary rows and remaining blanks. In this representative run, the selected skill improves SpreadsheetBench held-out test performance from 40.4 to 78.9.

F.7 Headline Numbers

For convenience, the headline aggregates over Table 1 are: (i) $52/52$ cells best or tied-best; (ii) average per-model improvement $\approx +17.6$ points across the seven direct-chat target models; (iii) average GPT–5.5 improvement of $+23.5$ (direct chat), $+24.8$ (Codex), $+19.1$ (Claude Code) over no skill; (iv) GPT–5.5 oracle-baseline gap of $+5.4$ points (direct chat) computed as the difference between SKILLOPT's six-benchmark average (82.3) and an oracle that picks the best of six competing methods *per cell* (76.9).

G Relation to Automatic Prompt Optimization and ProTeGi

ProTeGi [15] introduced the failure-conditioned textual-gradient loop: apply the current prompt to a batch, critique the errors in natural language, edit the prompt, evaluate the candidates, and iterate. We do not claim this generic loop as new. Later work such as TextGrad and GEPA, both of which we use as baselines, builds on the same idea. Our contribution is an optimization *control layer* for one persistent, deployable skill under expensive multi-turn rollouts, together with an empirical study of its behavior. Table 7 summarizes the mechanism-level differences, and Table 8 reports a matched head-to-head comparison (the same SKILLOPT and ProTeGi numbers also appear in Table 1).

	ProTeGi	**SKILLOPT**
Optimized object	A short instruction for a single-turn classifier	A persistent, versioned skill document deployed with a frozen agent
Update	Regenerated wholesale each round	Bounded add/insert/replace/delete edits on the current skill
Step size	Chosen implicitly by the LLM	Number of edits L_t, an explicit textual learning rate with schedules
Acceptance	Beam ranked by training-set score	Parent-relative test on a held-out selection split, with rollback
Rejected proposals	Discarded	Kept as negative feedback in an epoch-local buffer
Optimizer state	The beam	Rejected-edit buffer, slow update, optimizer-side meta skill
Evaluation	Four single-turn classification tasks	Six mostly multi-turn agentic benchmarks, seven targets, three harnesses
Transfer	Not studied	Cross-model, cross-harness, and nearby cross-benchmark

Table 7: Mechanism-level comparison between ProTeGi and SKILLOPT.

Target	**Method**	**SearchQA**	**Spreadsheet**	**OfficeQA**	**DocVQA**	**LiveMath**	**ALFWorld**
GPT–5.5	Initial skill	78.6	33.2	46.5	88.5	49.6	88.1
	ProTeGi	86.4	58.2	44.2	90.9	56.8	94.8
	SKILLOPT	**87.3**	**80.7**	**72.1**	**91.2**	**66.9**	**95.5**
GPT–5.4-nano	Initial skill	58.1	26.4	14.5	53.7	28.0	49.5
	ProTeGi	72.0	33.6	8.1	67.7	25.6	66.4
	SKILLOPT	**74.8**	**42.5**	**50.0**	**80.2**	**27.2**	**69.4**

Table 8: Matched comparison with ProTeGi [15]. Both optimizers start from the same initial skill and use the same splits, targets, and evaluators. Red marks cells where ProTeGi ends below its initial skill.

For the head-to-head comparison in Table 8, we ran ProTeGi with the same splits, target models, evaluators, and initial skill as SKILLOPT. We kept ProTeGi's six-round procedure and released meta-

prompts and only changed their wording to refer to an agent skill document. ProTeGi is competitive on short single-turn tasks. The gap widens on tool-intensive tasks, and in three cells ProTeGi finishes below its own initial skill: OfficeQA for both targets and LiveMath for GPT–5.4-nano. In those cells, optimization damaged a working artifact, which is the outcome that the parent-relative gate is designed to prevent.

H Three-Seed Results and Additional Model Families

Table 9 repeats SKILLOPT with three seeds (42/43/44) on two GPT targets and adds three Claude models and Qwen3.5–122B as targets. GPT–5.5 is the optimizer in every run, so the comparison across target models is controlled. For SearchQA, SpreadsheetBench, and LiveMath, the no-skill baselines are the single-run values used in the main protocol. For DocVQA, OfficeQA (with oracle parsed pages), and ALFWorld, the baselines were re-run with three seeds under this protocol, so they can differ from the single-run entries in Table 1.

Model		SearchQA	Spreadsheet	LiveMath	DocVQA	OfficeQA	ALFWorld
GPT–5.5	No skill	77.7	41.8	37.6	89.8±0.1	54.2±0.8	88.1±1.1
	SKILLOPT	86.7±0.6	80.0±0.6	66.4±0.5	91.1±0.5	67.2±1.7	90.3±0.6
GPT–5.4-mini	No skill	75.9	36.1	14.7	85.1±0.7	32.0±0.8	69.9±1.3
	SKILLOPT	81.9±1.5	64.5±5.1	34.1±1.1	89.3±1.0	50.1±5.2	80.9±2.5
Claude Opus 4.8	No skill	77.9	40.0	34.7	93.9±0.5	55.0±1.5	91.0±0.6
	SKILLOPT	84.0±0.5	67.4±3.9	39.2±1.7	95.5±0.5	60.7±3.7	94.3±0.4
Claude Sonnet 4.6	No skill	78.4	43.2	33.1	92.6±0.5	56.1±1.1	84.3±1.2
	SKILLOPT	85.0±0.5	66.9±1.8	42.7±6.3	93.1±0.3	56.6±1.7	86.1±1.9
Claude Haiku 4.5	No skill	70.1	38.6	29.0	66.0±0.3	39.0±1.1	64.9±0.6
	SKILLOPT	80.1±0.4	49.8±0.5	41.7±5.5	84.2±1.1	39.8±1.1	78.1±5.2
Qwen3.5–122B	No skill	76.4	41.1	26.6	92.7±0.9	42.8±2.0	63.7±1.5
	SKILLOPT	84.8±1.1	51.8±3.1	29.0±3.2	92.5±0.7	47.2±1.1	73.9±0.6

Table 9: Three-seed results (mean ± standard deviation) across six target models from three families. 35 of the 36 cells improve over no skill. The exception is Qwen3.5–122B on DocVQA, where the change (-0.2) is within one standard deviation of a 92.7 baseline.

Two patterns hold in every family. First, the gain grows with how much reusable procedure a task contains. SpreadsheetBench, which consists mostly of repeatable multi-step operations, has the largest gain for every target ($+10.7$ to $+38.2$). Second, the gain shrinks as the starting point gets stronger. Claude Opus gains $+27.4$ on SpreadsheetBench from a baseline of 40.0 but only $+1.6$ on DocVQA from 93.9, while Claude Haiku gains $+18.3$ on DocVQA from 66.0. The same holds within model sizes. Using Table 1 for Qwen3.5–4B and Qwen3.6–35B-A3B and Table 9 for Qwen3.5–122B, the mean gain is $+19.2$, $+9.1$, and $+6.0$, respectively. Across GPT–5.4-nano, mini, and full, it is $+26.7$, $+15.4$, and $+12.7$. A skill can close part of the gap to the next size tier. With an optimized skill, Qwen3.5–4B reaches 81.3 on ALFWorld, compared with 63.7 for Qwen3.5–122B without one. GPT–5.4-nano with a skill matches or exceeds GPT–5.4 without one on SpreadsheetBench, OfficeQA, and DocVQA. We do not claim that a skill is equivalent to a larger model.

I Non-Verifiable and Noisy Rewards

LLM-judged benchmarks. WritingBench [44] and SlidesBench [45] have no programmatically checkable answer and are scored by an LLM judge. GPT–5.5 serves as both optimizer and target, with three seeds (Table 10). On SlidesBench, the continuous soft score moves in the same direction as the hard score ($66.85\pm0.64 \rightarrow 78.46\pm1.86$), so the gain does not come from a hard-metric threshold.

Benchmark	Judge	No skill	SKILLOPT	Δ
WritingBench	Claude Sonnet 4.6	95.56±1.57	98.33±0.00	$+2.78 \pm 1.57$
SlidesBench	GPT–5.4-mini	71.11±1.47	83.59±1.92	$+12.48 \pm 1.89$

Table 10: SKILLOPT with LLM-judged rewards (three seeds).

Label noise. On SearchQA, with GPT–5.5 as both optimizer and target, we flip 10%, 20%, or 30% of item-level labels. The flips are deterministic and are applied through two independent channels:

the rollout feedback used during training and the reward used by the selection gate. The 1,400-item test split is always scored with clean labels, and gold answers are never shown to the optimizer. Each condition uses three seeds and 40 optimizer steps, for 24 runs in total (Table 11). Every run retains at least $+6.31$ points over the untrained skill, and performance does not degrade monotonically up to 30% corruption in either channel. With three seeds, the differences between noise levels are descriptive only.

Condition	Rollout noise	Gate noise	Clean-test EM	Δ vs. untrained
Initial skill, untrained	–	–	78.52±0.12	–
Clean training reward	0%	0%	85.76±0.12	$+7.24 \pm 0.17$
Rollout noisy	20%	0%	85.98±0.99	$+7.45 \pm 0.88$
Gate noisy	0%	20%	84.83±0.74	$+6.31 \pm 0.86$
Both noisy	20%	20%	85.86±0.53	$+7.33 \pm 0.43$
Rollout channel, 10% / 30%	–		85.31±0.71 / 86.12±0.28	
Gate channel, 10% / 30%	–		85.33±0.40 / 85.81±0.93	

Table 11: SearchQA with corrupted training rewards (three seeds per condition).

J Retrieval Memory and Fine-Tuning Controls

Retrieval memory. We built a mem0 [46] memory from the same training pool by LLM fact extraction and used top-3 retrieval with GPT–5.5. Because this is a sampled diagnostic, the evaluation sample sizes are small (Table 12). Retrieval is competitive on fact-oriented SearchQA but falls below no skill on SpreadsheetBench and LiveMath. Retrieval suits changing instance-level facts or episodes, whereas SKILLOPT distills stable rules for how to act and verify. The two approaches can be complementary.

Method (GPT–5.5)	SearchQA	SpreadsheetBench	LiveMath
No skill	77.7	41.8	37.6
mem0 retrieval (top-3)	85.7 (n=14)	14.3 (n=28)	20.0 (n=25)
SKILLOPT (three seeds)	**86.7±0.6**	**80.0±0.6**	**66.4±0.5**

Table 12: Comparison with mem0 retrieval memory.

Weight adaptation. For Qwen3.5–4B, we fine-tuned all parameters for one epoch on the model's own successful training-split rollouts (Table 13). SKILLOPT is higher on all six benchmarks. This SFT control learns only from successful trajectories, whereas SKILLOPT also turns scored failures into procedural corrections. SKILLOPT is therefore most useful when weights are inaccessible, successful demonstrations are scarce, or one human-readable artifact should be reused across compatible deployments. Fine-tuning is attractive when weights are accessible, successful data is abundant, and the deployment target is fixed.

Qwen3.5–4B	SearchQA	Spreadsheet	OfficeQA	DocVQA	LiveMath	ALFWorld
No skill	68.1	9.3	14.5	86.9	22.4	30.6
Full-parameter SFT	70.1	10.4	15.7	88.5	23.2	35.1
SKILLOPT	**71.2**	**23.9**	**29.7**	**89.0**	**52.0**	**81.3**

Table 13: One-epoch full-parameter SFT on Qwen3.5–4B, trained on its own successful training-split rollouts.

K Transfer Analysis

Shared semantic core across harnesses. To understand why cross-harness transfer works, we compared four independently optimized SpreadsheetBench skills as a post-hoc diagnostic. They share ten operational rules, including structure-first inspection, table localization, literal-value materialization, and post-save verification. We kept only this shared core, rendered it in each source's

wording, and evaluated both renderings with GPT–5.5 fixed (Table 14). Both renderings improve both harnesses, which indicates that independently learned task-level rules remain actionable under a different execution loop.

Evaluation harness	No skill	Shared core, Codex wording	Shared core, Claude Code wording
Codex	39.3	89.3	78.6
Claude Code	53.6	75.0	75.0

Table 14: Shared-core SpreadsheetBench skill evaluated in both harnesses (GPT–5.5, diagnostic subset).

Illustrative wrong→right transfers. In each of the following cases, the target agent and task are fixed, and only its own skill is replaced by the skill learned in the other harness. These cases are mechanism illustrations, not additional quantitative cells. They use GPT–5.5 on Codex and Claude Sonnet 4.6 on Claude Code. (i) *Codex→Claude Code, SpreadsheetBench:* the target wrote an unevaluated formula whose cached value was `None`. The transferred rule says to compute in Python and write literal values. (ii) *Claude Code→Codex, SpreadsheetBench:* the target omitted part of a required aggregate. The transferred rules require all relevant rows, contiguous output, and cleanup of stale cells. (iii) *Codex→Claude Code, LiveMath:* the answer changes after the transferred rule asks the model to rank viable answers by logical strength. (iv) *Claude Code→Codex, LiveMath:* the answer changes after the transferred skill warns that an exact asymptotic expression is an extra claim that must be rejected unless it is proved.

What is transferred. We classified learned incremental rules into four categories. **P** covers general reasoning procedures, such as canonicalizing header keys or treating "contains" as case-insensitive substring matching. **T** covers tool-use strategies, such as reading cached formula values from a second workbook loaded with `data_only=True` and verifying after saving that cells contain scalars rather than `None`. **O** covers output and range constraints, such as clearing only the destination range or treating totals and headers separately. **B** covers narrow benchmark-pattern instructions, such as a specific residual-balancing order. We removed one category at a time and evaluated cross-model transfer on 140 held-out SpreadsheetBench items per direction (Table 15). Removing B or O changes accuracy little, whereas removing T causes the clearest drops. Tool-use knowledge such as formula materialization, cached-value reading, and output verification is therefore an important transferable component, and the transfer gains are not explained mainly by formatting or narrow benchmark heuristics. These are single runs, and some removals slightly *improve* accuracy, which suggests redundancy between rules and interactions among instructions. We therefore treat the category differences as preliminary and descriptive.

Source skill → target model	Full	no-P	no-T	no-O	no-B
GPT–5.5 → Claude Sonnet 4.6	71.4	77.9	64.3	72.9	72.9
Claude Sonnet 4.6 → GPT–5.5	73.6	72.1	47.1	75.7	70.0

Table 15: Category ablation of transferred SpreadsheetBench skills (hard accuracy, 140 held-out items per direction, single run).

How cross-benchmark transfer fails. Table 3 applies source skills unchanged, with no format alignment or post-processing. In separate probe runs, we also evaluated a variant whose output-format instructions were aligned to the target benchmark (Table 16). Baselines come from these probe runs and can differ slightly from Table 3. These probes show three failure modes. *Format collision:* raw transfer is negative but becomes positive once the answer format is aligned, so the artifact carries a transferable procedure together with a non-transferable output convention. *Capacity:* on the weakest model, alignment hurts, because extra formatting constraints consume capacity that the model needs for the task. *Reasoning redirection:* on paired GPT–5.4 OlympiadBench generations that differ only in whether the LiveMath skill is injected, two items were answered correctly without the skill but incorrectly with it, and both responses had valid boxed answers. In one, the transferred skill led the model to prune a valid case without proof. In the other, it led the model to mis-enumerate a counting set. A skill specialized for theorem-level multiple-choice reasoning can redirect search without supplying the exhaustive checks that an open-ended problem needs.

Source → target (model)	No skill	Raw transfer	Format-aligned transfer
LiveMath → OlympiadBench (GPT–5.4)	62.7	61.9 (−0.8)	64.8 (+2.1)
OlympiadBench → Omni-MATH (GPT–5.4)	57.6	56.2 (−1.4)	58.7 (+1.1)
OlympiadBench → Omni-MATH (GPT–5.4-nano)	38.8	40.1 (+1.3)	35.0 (−3.8)

Table 16: Cross-benchmark format-alignment probes.

Using transferred skills. The evidence supports four practical guidelines, all scoped to the settings we tested. (1) The most reliable regime keeps the task fixed and changes the model or harness. Within the tested Codex–Claude Code pair, transfer improved over no skill in both directions on both benchmarks, but we have not tested other harness pairs. (2) Align the output format before reusing a skill on a different benchmark. (3) Transferring to weaker targets yields the largest gains, but very weak targets can be hurt by over-specified skills. (4) Re-validate on a small target-side held-out set before deployment. This check is cheap relative to re-optimization and would catch every failure case above. Universal transfer across unrelated domains is outside the intended scope of a skill, which is a carrier of task-specific procedure for fast domain adaptation.

L Reproducibility Details

Splits and selection budget. Table 17 lists the split sizes and the maximum number of selection sweeps. One sweep evaluates every item in the selection set once. With $E{=}4$ epochs, accumulation $A{=}1$, and rollout batch B, the scheduled upper bound is $Q_{\max} = 1 + E\lceil N_{\text{train}}/(BA)\rceil + (E-1)$. The first term evaluates the initial skill, the second evaluates step candidates, and the third allows at most one slow-update candidate at each of the last three epoch boundaries. Hash caching, repeated candidates, and empty candidates can only reduce these counts. The test split never participates in proposal, acceptance, or skill selection.

Task	Train / selection / test	B	Max sweeps $Q_{\max}$	Max selection-item evaluations
SearchQA	400 / 200 / 1,400	40	44	8,800
SpreadsheetBench	80 / 40 / 280	40	12	480
OfficeQA	50 / 24 / 172	40	12	288
DocVQA	107 / 53 / 374	40	16	848
LiveMath	35 / 17 / 125	200	8	136
ALFWorld	39 / 18 / 134	80	8	144

Table 17: Split sizes and selection-query budget. A single item can require multiple target-model calls.

Optimization cost. Table 6 reports the total training tokens, the tokens per absolute test point, and the final skill length for each GPT–5.5 case study. Table 18 separates the cost of three four-epoch GPT–5.5/GPT–5.5 ALFWorld runs by role. Because each reflection and update step batches many trajectories, the optimizer accounts for only 0.7% of model calls and 4.9% of tokens.

Target calls	Optimizer calls	Target tokens	Optimizer tokens	Total tokens	Wall-clock
5,848.3±299.5	41.7±0.5	12.15±0.83M	0.62±0.03M	12.77±0.86M	4.56±0.41 h

Table 18: Optimization cost by role for four-epoch ALFWorld optimization (GPT–5.5 target and optimizer, seeds 42/43/44).

Baseline resources. All cells in Table 1 were produced under one aligned protocol. Runs that could not be aligned are shown as "–" and are not mixed into the table. Every method uses the same frozen target and backend, task interface, deterministic splits, and evaluator. Table 19 summarizes how the remaining resources are controlled.

Prompts, schemas, and release. Appendix E.2 reproduces the optimizer and reflection prompts with their required JSON outputs. Appendix E.3 specifies the guarded `append`, `insert_after`,

Method	Rollout evidence	Validation access	Optimizer / teacher	Horizon / budget
No / human / LLM skill	None	None	One-shot generator: GPT–5.5 (LLM skill)	–
Trace2Skill	Same training pool	Native rule, matched budget	GPT–5.5	Matched
TextGrad	Same training pool	Native rule, matched budget	GPT–5.5	Matched
GEPA	Same training pool	Native rule, matched budget	GPT–5.5	Matched
EvoSkill	Same training pool	Native rule, matched budget	GPT–5.5	Matched
ProTeGi	Same training pool	Native beam selection	GPT–5.5	Six rounds
SKILLOPT	Same training pool	Selection split, $\leq Q_{\max}$ sweeps	GPT–5.5	Four epochs

Table 19: Resources available to each method. "Matched" means the same optimization horizon, reasoning effort, maximum completion-token limit, and data and rollout budget, while each method keeps its native update procedure and hyperparameters. No method uses test feedback for selection.

`replace`, and `delete` operations. Optimization uses a fixed four-epoch budget with no data-dependent early stopping and exports the skill with the best selection score. We will release the complete code, raw and rendered prompt templates, task configurations, seeds, evaluation scripts, and the skills before and after optimization for all reported experiments at https://aka.ms/SkillOpt.

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: The abstract and introduction state the contribution (a controllable text-space optimizer for one persistent skill, plus an empirical study) and credit the underlying execute–critique–edit loop to prior prompt optimization (Section 2). The claims are supported in Section 4 and Appendices H–L, and transfer is scoped to the tested models, harnesses, and nearby benchmarks (Section 4.3).

   Guidelines:

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

2. **Limitations**

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

   Answer: [Yes]

   Justification: Appendix B discusses the reliance on reliable automatic feedback and a held-out selection split, the one-time optimization cost, the single-skill design, and the need to re-validate skills before transfer. Section 4.3 and Appendix K state the scope and failure modes of transfer.

   Guidelines:

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

3. **Theory assumptions and proofs**

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

   Answer: [N/A]

   Justification: The paper presents no formal theoretical results; the optimization procedure is specified algorithmically (Section 3, Algorithm 1) and evaluated empirically.

   Guidelines:

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

4. **Experimental result reproducibility**

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

   Answer: [Yes]

   Justification: Section 3 and Algorithm 1 specify the optimization loop, and Appendices E, E.2, E.3, and L give the benchmarks, hyperparameters, optimizer prompts and edit schemas, split sizes, selection budgets, and baseline resources. All models are named with their versions; reproducing the closed-model results requires API access to the same versions.

   Guidelines:

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

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

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

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

   Answer: [Yes]

   Justification: Code, optimizer prompt templates, per-benchmark configurations, split manifests, and reference skills are publicly available at `https://aka.ms/SkillOpt` (MIT license) with instructions for installation, data preparation, training, and evaluation; benchmark data are obtained from their public sources. The remaining artifacts listed in Appendix L (per-experiment configurations, seeds, and skills before and after optimization) will be added to the same repository.

   Guidelines:

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

6. **Experimental setting/details**

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

   Answer: [Yes]

   Justification: Section 4 and Appendix E give the benchmarks, metrics, baselines, and default hyperparameters, and Appendix L lists split sizes, per-benchmark batch sizes, selection budgets, and baseline resources (Tables 17 and 19). Sensitivity to the main hyperparameters is reported in Appendix F.2.

   Guidelines:

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

7. **Experiment statistical significance**

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

Answer: [Yes]

Justification: Appendices H and I and Table 18 report mean ± one standard deviation over three seeds (42/43/44) for six target models, the LLM-judged and noisy-reward studies, and the cost accounting. Table 1, the ablations, the ProTeGi comparison, and the transfer results are single runs because of the cost of repeating closed-model and harness-backed experiments.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

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

   Answer: [Yes]

   Justification: Because the GPT and Claude models are accessed through APIs, compute is reported as model calls, tokens, and wall-clock time (Tables 18, 6, and 17). The hardware used for the open-weight Qwen runs and the compute of preliminary runs are not reported.

   Guidelines:

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

9. **Code of ethics**

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

   Answer: [Yes]

   Justification: The work uses public benchmarks and commercial or open-weight models under their terms of use, involves no human subjects or personal data, and conforms to the NeurIPS Code of Ethics.

   Guidelines:

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

10. **Broader impacts**

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

    Answer: [Yes]

    Justification: Appendix D discusses positive impacts (auditable adaptation of frozen agents without weight updates) and negative ones (encoding harmful procedures in a skill or exploiting evaluator flaws), together with mitigations.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

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

    Answer: [N/A]

    Justification: The paper releases no pretrained model, generator, or scraped dataset; the released code, prompts, and natural-language skills for public benchmarks pose no high risk of misuse.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.

- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

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

    Answer: [No]

    Justification: All benchmarks, baselines, harnesses, and models are credited by citation or by provider and version and were used under their licenses or terms of use, but the paper does not list the license of each asset.

    Guidelines:

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

13. **New assets**

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

    Answer: [Yes]

    Justification: The released code and learned skills (`https://aka.ms/SkillOpt`, MIT license) are documented in the repository and in the paper (Section 3, Appendices E.2, E.3, and F.5); they contain no data collected from people.

    Guidelines:

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

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

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

    Answer: [N/A]

    Justification: The paper involves no crowdsourcing or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

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

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

    Answer: [N/A]

    Justification: The paper involves no human subjects, so IRB approval or an equivalent review was not required.

    Guidelines:

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

16. **Declaration of LLM usage**

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

    Answer: [Yes]

    Justification: LLMs are core components of the method: a frozen target LLM executes tasks and a separate optimizer LLM proposes skill edits (Section 3). LLMs also produce the one-shot LLM-skill baseline, serve as judges for WritingBench and SlidesBench (Appendix I), and extract facts for the mem0 control (Appendix J).

    Guidelines:

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