SKILLADAM: STABLE AND EFFICIENT SKILL EVOLUTION FOR AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Agent skills provide a lightweight mechanism for equipping frozen language-model agents with domain knowledge and procedural guidance. However, building high-quality skills remains costly and difficult to scale. As expert-written skills require substantial human effort, recent skill self-evolution methods automate skill optimization using execution feedback. However, they still rely on heuristic update strategies that often lead to unstable optimization and low iteration efficiency. To address these limitations, stable optimization should follow two key principles. (1) *Direction Stability* requires effective corrections to accumulate across iterations rather than be overwritten by iteration-local feedback. (2) *Update Adaptivity* requires the scope of each revision to reflect the consistency of recent case-level improvements. We introduce **SKILLADAM**, an Adam-inspired framework that realizes these optimization principles for optimizing discrete and non-differentiable skill documents. Inspired by Adam's first moment, SKILLADAM maintains an optimization memory that accumulates identified problems and prior solution attempts to stabilize the update direction. Inspired by Adam's second moment, it maintains a volatility-driven edit budget that tracks the history-weighted variation of recent case-level improvements to adaptively control the update magnitude. Extensive experiments on seven benchmarks spanning both short- and long-horizon tasks show that SKILLADAM achieves state-of-the-art performance with substantially more stable optimization, while requiring significantly fewer optimization iterations and lower optimization cost than prior methods.[1]

1 INTRODUCTION

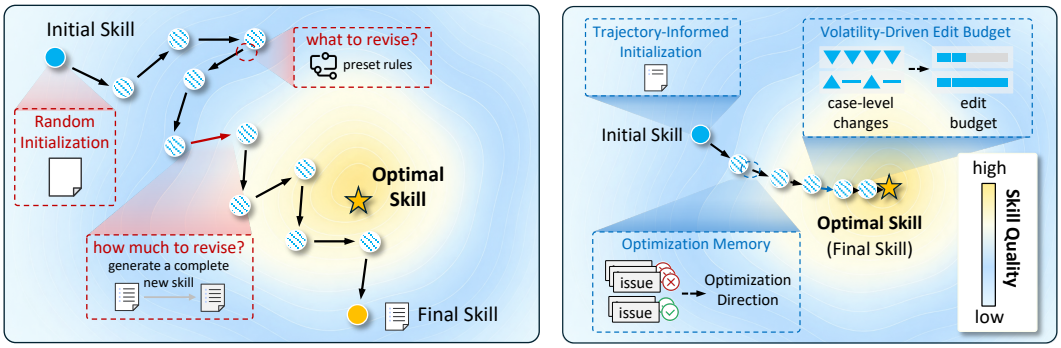
Large language model (LLM) agents solve complex real-world tasks through multi-step planning and tool use. Representative tasks include web interaction (Wang et al., 2025a), travel planning (Zhang et al., 2026b), data preparation (Fan et al., 2026; Deng et al., 2026), and data discovery and analysis (Zhang et al., 2025; 2026c; Liu et al., 2026). However, their general-purpose capabilities often fall short in domain-specific scenarios that require specialized knowledge and procedural guidance. For example, planning a multi-city trip requires reasoning about visa regulations, transportation dependencies, and airline-specific booking policies. To address this limitation, recent agent frameworks introduce *Agent Skills*, defined by Anthropic as modular packages of instructions and supporting resources that equip agents with specialized capabilities (Anthropic, 2025).

Obtaining high-quality Skills remains a fundamental challenge. SkillsBench shows that carefully curated Skills can substantially improve agent performance (Li et al., 2026), while SkillAxe reports that Skills generated directly by LLMs often remain ineffective without further refinement (Gautam et al., 2026). Domain experts can manually author Skills, but this process is time-consuming and labor-intensive. Language models reduce this burden by generating or refining Skills, yet their heavy reliance on human-designed prompts and task-specific heuristics limits scalability across diverse domains.

To reduce human intervention, recent studies have explored *automated Skill construction* from interaction experience. AutoManual incrementally updates structured rules and compiles them into an instruction manual (Chen et al., 2024). Agent Skill Induction learns verified programmatic Skills

[1] Code are available in supplementary materials.

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



(a) Existing SGD-like methods.

(b) SKILLADAM (ours).

Figure 1: Conceptual comparison of iterative skill optimization strategies.

from web interactions (Wang et al., 2025a), while Trace2Skill synthesizes a unified Skill from a diverse pool of execution traces (Ni et al., 2026). More recently, skill self-evolution has emerged as a dominant paradigm for automated Skill construction. Some methods iteratively refine a Skill document (Gautam et al., 2026; Yang et al., 2026), while others evolve Skill packages or maintain Skill repositories (Alzubi et al., 2026; Zhang et al., 2026a; Ouyang et al., 2026). Although these methods reduce manual effort through iterative refinement, they still rely on heuristic update strategies. As illustrated by the iteration-local optimization path in Figure 1(a), existing methods still lack effective control over both the revision direction and the revision extent in each iteration.

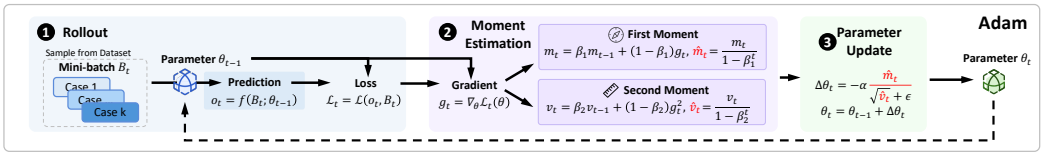
To address this limitation, we propose **SKILLADAM**, a stable and efficient framework that aims to preserve effective corrections across iterations while adapting each revision to the reliability of recent evidence. Realizing these properties raises two challenges. The first is **Direction Stability**. Since each iteration observes feedback from only a limited set of cases, successive revisions may focus on different problems and undo one another. For example, one revision may instruct the agent to minimize the total cost by selecting the cheapest feasible itinerary. A later revision may add more attractions to produce a richer travel plan, but these additions can increase the cost and violate the earlier budget constraint. New revisions must therefore remain consistent with effective corrections accumulated in earlier iterations. The second challenge is **Update Adaptivity**. The appropriate edit scope should depend on how consistently a recent revision affects the evaluated cases. If one revision improves some cases but degrades others, a broad subsequent edit risks overwriting useful guidance and should therefore be constrained. If the gains are consistent across cases, a broader edit is better supported.

These challenges resemble those in stochastic optimization, where each update is based on partial evidence and the appropriate step size depends on the scale of recent signals. Adam (Kingma & Ba, 2015) provides two complementary design principles: aggregate historical update signals to stabilize direction, and rescale the effective step size using accumulated signal magnitude. SKILLADAM adopts these principles functionally in the discrete skill space rather than applying Adam numerically.

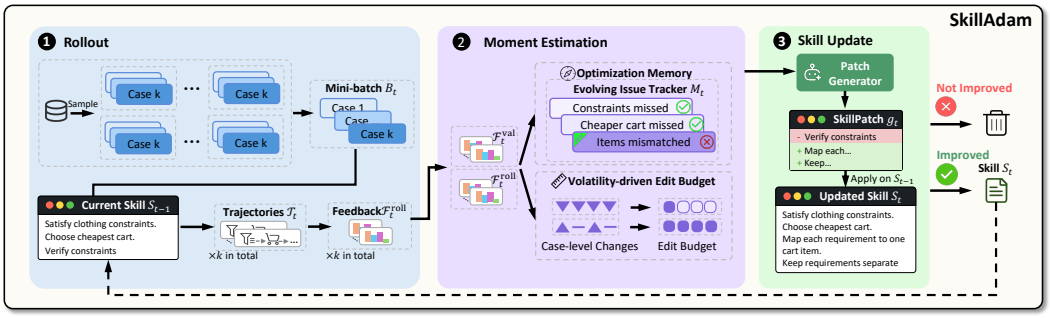
As illustrated in Figure 1(b), SKILLADAM first constructs a trajectory-informed initial skill from execution trajectories and their evaluation feedback. During self-evolution, an *Evolving Issue Tracker* records identified problems, their current status, and the outcomes of prior solution attempts, allowing each new update to consider both current feedback and accumulated evidence. In parallel, a *volatility-driven edit budget* measures how unevenly a candidate changes case-level performance and controls the allowable scope of the next modification. The tracker determines what should be revised, while the budget determines how much may be revised. Together with trajectory-informed initialization, these components produce the more coherent and efficient optimization path shown in Figure 1(b).

Our contributions are summarized as follows:

- We propose **SKILLADAM**, an Adam-inspired framework for stable and efficient skill self-evolution.



(a) Adam inspiration. Adam aggregates first- and second-moment estimates to stabilize the update direction and adapt the effective step size.



(b) SKILLADAM framework. SKILLADAM uses the Evolving Issue Tracker and a volatility-driven edit budget to provide analogous control over the direction and magnitude of skill updates in the discrete skill space.

Figure 2: Functional correspondence between Adam and SKILLADAM.

- We introduce an *optimization memory* and a *volatility-driven edit budget* as functional analogues of Adam's first- and second-moment mechanisms, respectively stabilizing the optimization direction and adapting the update magnitude.
- We conduct extensive experiments on seven benchmarks, where SKILLADAM achieves state-of-the-art performance while requiring substantially fewer optimization iterations and reducing overall optimization cost.

2 RELATED WORK

Agent Skills. We follow Anthropic's official definition of *Agent Skills* as modular packages of instructions and supporting resources that equip an agent with specialized capabilities (Anthropic, 2025). Because Skills are stored independently of model parameters, the same package can be distributed to compatible agents without retraining. Earlier work had already externalized reusable capabilities, although it did not share this package definition. Voyager stores executable programs in a library that can be retrieved for later tasks (Wang et al., 2023). AutoManual learns structured rules through interaction and compiles them into a readable instruction manual (Chen et al., 2024). Agent Skill Induction learns and verifies programmatic skills for web agents (Wang et al., 2025a). SkillsBench provides a common evaluation of package-based Agent Skills and shows that curated Skills improve agent performance across diverse tasks (Li et al., 2026). ExpeL and Agent Workflow Memory instead retain natural-language experience or reusable workflows from past executions (Zhao et al., 2023; Wang et al., 2025b). Our work focuses on Skills whose primary artifact is a natural-language instruction document and studies their iterative optimization under task evaluation.

Prompt Optimization. Automatic prompt optimization studies how to improve discrete language artifacts without updating model parameters. OPRO proposes new instructions from previously evaluated candidates and their scores (Yang et al., 2024). ProTeGi turns error feedback into textual gradients and applies search to select prompt edits (Pryzant et al., 2023). TextGrad propagates natural-language feedback through a computation graph to optimize prompts and other textual components (Yuksekgonul et al., 2024). GEPA uses reflective feedback from rollouts to evolve prompts (Agrawal et al., 2026). ERM retains feedback from earlier attempts to support exemplar-guided prompt optimization (Yan et al., 2025). They show that evaluation signals can guide discrete textual updates without changing model parameters. Their optimization targets are prompts or com-

ponents of language-model programs, while SKILLADAM optimizes a reusable Agent Skill used across task instances.

Skill Self-Evolution. Recent work directly automates Agent Skill construction and revision. Trace2Skill analyzes a broad pool of executions and consolidates trajectory-local lessons into a unified skill directory (Ni et al., 2026). SkillAxe iteratively diagnoses and refines LLM-authored skill documents with structured evaluation signals (Gautam et al., 2026). EvoSkill discovers and revises skills through failure analysis, then retains validated candidates through Pareto selection (Alzubi et al., 2026). CoEvoSkills jointly evolves a Skill Generator and a Surrogate Verifier to construct multi-file Skill packages without ground-truth test content (Zhang et al., 2026a). SkillOS trains a curator that updates an external skill repository from accumulated experience (Ouyang et al., 2026). SkillOpt is closest to our setting because it applies bounded textual edits to one skill document and accepts an update only when validation performance improves (Yang et al., 2026). Their optimization targets differ. Some optimize one document, while others construct packages or maintain a repository. Across these settings, existing methods do not jointly maintain persistent optimizer states for the direction and magnitude of successive revisions.

3 METHOD

To enable stable and efficient skill self-evolution in a discrete skill space, we propose SKILLADAM with two Adam-inspired mechanisms: an Evolving Issue Tracker that stabilizes the update direction and a volatility-driven edit budget that adapts the update magnitude. We describe the framework and its components below.

3.1 PROBLEM FORMULATION

Let $\mathcal{S}$ be the discrete space of skills and $\mathcal{D}$ a task dataset. A skill $S \in \mathcal{S}$ is a Markdown-formatted natural-language instruction document that guides an agent in a target domain. For each task $d \in \mathcal{D}$, the evaluator returns $\mathcal{E}(S, d) = (E(S, d), C(S, d))$, comprising a metric vector E and optional diagnostic information C. We write $\mathcal{F}(S, B)$ for the case-level feedback on a batch B. A task-dependent function Φ aggregates the numerical metrics into the optimization objective:

$$J(S) = \Phi(\{E(S, d) \mid d \in \mathcal{D}\}), \qquad S^* = \arg\max_{S \in \mathcal{S}} J(S). \tag{1}$$

Diagnostic information remains available to the language-based update and memory functions. Expanded feedback definitions appear in Appendix C.

3.2 THE SKILLADAM FRAMEWORK

Framework Overview. Figure 2(b) presents the SKILLADAM cycle, organized into *rollout*, *moment estimation*, and *skill update*. Before iterative optimization begins, SKILLADAM constructs an initial skill S_0 from execution trajectories and their evaluation feedback. At iteration t, the current skill is executed on a sampled mini-batch, and the optimizer states carried from the previous iteration, M_{t-1} and σ_t, guide the generation of a candidate modification. After candidate evaluation, the paired rollout and validation feedback updates the Evolving Issue Tracker and the improvement-volatility estimate.

The three blocks describe functional roles rather than a strict within-iteration execution order. The memory and volatility states obtained from iteration t guide the skill update at iteration $t+1$. Algorithm 1 in Appendix C.3 gives the complete operational order.

Rollout. At iteration t, SKILLADAM samples a mini-batch $B_t \subset \mathcal{D}$ and executes the current skill S_{t-1}, collecting trajectories and case-level feedback:

$$\mathcal{T}_t = \mathrm{Rollout}(S_{t-1}, B_t), \qquad \mathcal{F}_t^{\mathrm{roll}} = \mathcal{F}(S_{t-1}, B_t). \tag{2}$$

Here, $\mathcal{F}(S, B)$ collects the evaluator outputs $\mathcal{E}(S, d)$ for all $d \in B$, including metric scores and diagnostic information; Appendix C.1 gives the formal definition. The trajectories record how the agent behaves. Together, the trajectories and feedback provide the iteration-specific optimization signal used to generate the next skill update.

Moment Estimation. SKILLADAM maintains two persistent optimizer states: optimization memory M_t, instantiated as an *Evolving Issue Tracker*, and historical volatility V_t, which controls the magnitude of subsequent updates. Both states are refreshed after candidate evaluation and carried into the next iteration.

The tracker maintains error patterns, their current status, and previous solution attempts with observed outcomes. After the candidate has been evaluated, it evolves as

$$M_t = \mathcal{U}_{\mathrm{EIT}}(M_{t-1}, \mathcal{T}_t, \mathcal{F}_t^{\mathrm{roll}}, g_t, \mathcal{F}_t^{\mathrm{val}}, a_t). \tag{3}$$

The LLM-based update links failures to existing issues or creates new entries, records each attempted modification and its outcome, and reopens recurring failures. Retaining these histories lets subsequent revisions build on accumulated evidence and stabilizes the update direction.

SKILLADAM also estimates how consistently the candidate affects the evaluated cases. For case $d_i \in B_t$, let $\mathcal{E}_{t,i}^{\mathrm{roll}}$ and $\mathcal{E}_{t,i}^{\mathrm{val}}$ denote the current and candidate case-level feedback, respectively. The improvement is $\delta_{t,i} = s(\mathcal{E}_{t,i}^{\mathrm{val}}) - s(\mathcal{E}_{t,i}^{\mathrm{roll}})$, where s extracts the benchmark-specific scalar score. The sample variance $\widehat{V}_t$ of these improvements measures current volatility; it is set to zero when fewer than two valid comparisons are available. Starting from $V_0 = 0$, we update the historical volatility as $V_t = \beta_2 V_{t-1} + (1-\beta_2)\widehat{V}_t$. The next edit budget is

$$\sigma_{t+1} = \max\left(b_{\min}, \left\lfloor b_{\mathrm{base}} \left[1 - \mathrm{clip}\left(\frac{V_t}{V_{\max}}, 0, 1\right)\right]\right\rfloor\right). \tag{4}$$

Here, $\beta_2 \in [0, 1)$ is the EMA decay rate; b_{base}, $b_{\min}$, and $V_{\max}$ are the base budget, minimum budget, and volatility saturation threshold. The integer budget σ_t specifies the maximum number of unified-diff hunks requested from the patch generator, with each hunk intended to contain one logical change. It is a prompt-level constraint: patches are not rejected or truncated solely for exceeding the budget, and empty patches are allowed. Higher volatility reduces the requested edit scope. Appendix B.3 reports the benchmark-specific settings.

Skill Update. At iteration t, the LLM-based patch generator proposes a modification conditioned on the current rollout and the optimizer states carried from the previous iteration:

$$g_t = \mathcal{G}_{\mathrm{LLM}}(S_{t-1}, \mathcal{T}_t, \mathcal{F}_t^{\mathrm{roll}}, M_{t-1}, \sigma_t). \tag{5}$$

Applying g_t produces a candidate $\widetilde{S}_t = \mathrm{Apply}(S_{t-1}, g_t)$. It is evaluated on the same mini-batch to obtain $\mathcal{F}_t^{\mathrm{val}} = \mathcal{F}(\widetilde{S}_t, B_t)$, providing directly comparable case-level feedback. The acceptance gate requires at least one designated metric to satisfy its benchmark-specific gain threshold while all protected metrics remain within their regression bounds. Appendix B.3 lists the metrics and thresholds.

$$a_t = G_\theta(S_{t-1}, \widetilde{S}_t, \mathcal{F}_t^{\mathrm{roll}}, \mathcal{F}_t^{\mathrm{val}}) \in \{0, 1\}, \tag{6}$$

$$S_t = \begin{cases} \widetilde{S}_t, & a_t = 1, \\ S_{t-1}, & a_t = 0. \end{cases} \tag{7}$$

Both accepted and rejected evaluated attempts update the tracker and volatility state for the next iteration. If the patch cannot be applied, the skill, memory, volatility, and budget remain unchanged.

4 EXPERIMENTS

Benchmarks. We evaluate SKILLADAM on seven benchmarks. In our setting, *short-horizon* benchmarks require fewer than ten tool calls per task on average: SearchQA, SpreadsheetBench, OfficeQA Full, DocVQA, and LiveMathematicianBench (LMB; LiveMath in the tables). The *long-horizon* benchmarks, ALFWorld and DeepPlanning, require at least ten tool calls per task on average. Appendix B.1 provides a separate introduction to each benchmark.

Metrics. We report Exact Match for SearchQA, OfficeQA, and LiveMath; hard task success for SpreadsheetBench; ANLS-hard for DocVQA; episode goal-completion rate for ALFWorld; and case accuracy for DeepPlanning. All scores are percentages. Appendix B.1 describes the benchmarks and evaluation metrics, including the aggregation of DeepPlanning results.

4.1 BASELINES

We compare against NoSkill, HumanSkill, LLMSkill, Trace2Skill (Ni et al., 2026), TextGrad (Yuksekgonul et al., 2024), GEPA (Agrawal et al., 2026), and SkillOpt (Yang et al., 2026). Short-horizon baseline scores and ALFWorld NoSkill are taken from SkillOpt; we reproduce ALFWorld SkillOpt and both DeepPlanning baselines. Appendix B.2 details the baselines and identifies the source of each set of results. Within each benchmark, methods share the target-agent configuration, test cases, and evaluator. SkillOpt retains its native selection, slow-update, and optimizer-memory procedures and runs for four epochs.

4.2 SETUP

All runs use frozen target models in a no-harness, direct-chat setting: GPT-5.5 outside DeepPlanning and Claude Sonnet 4.5 on DeepPlanning. SKILLADAM and SkillOpt share the same initial skill, constructed from fixed baseline execution trajectories. For the six benchmarks outside DeepPlanning, both methods use the original non-test partitions and retain the same test split. SkillOpt uses selection scores throughout optimization to accept or reject regular updates and retain the best skill; score changes associated with rejected edits also inform subsequent proposals. SKILLADAM combines the train and selection partitions into one optimization pool, from which it samples cases for both revision generation and acceptance evaluation. This changes how the existing non-test examples contribute to optimization without introducing additional examples.

For the six benchmarks outside DeepPlanning, SKILLADAM processes sequential batches from a single traversal of the optimization pool, subject to the stopping settings in Appendix B.3. DeepPlanning uses odd case IDs for optimization and even IDs for testing; optimization batches are resampled at each iteration, with iteration limits and early stopping after consecutive unaccepted updates. Each candidate is compared with the current skill on the same sampled cases, without a separate validation set. SkillOpt runs for four epochs with its original split and native selection and slow-update rules. For our main-result runs, each final skill is evaluated once per test case. Figure 4 additionally reports post-hoc evaluations of intermediate candidates; test scores are not used to select updates.

4.3 MAIN RESULTS

Results on Short-Horizon Benchmarks. As shown in Table 1, SKILLADAM achieves state-of-the-art performance among the compared methods on short-horizon tasks in the GPT-5.5 no-harness setting.

Results on Long-Horizon Benchmarks. Table 2 shows stronger advantages for SKILLADAM on long-horizon benchmarks, particularly on DP-Travel, where NoSkill and SkillOpt nearly fail. Figure 3 summarizes a complementary skill-quality evaluation, detailed in Appendix D.

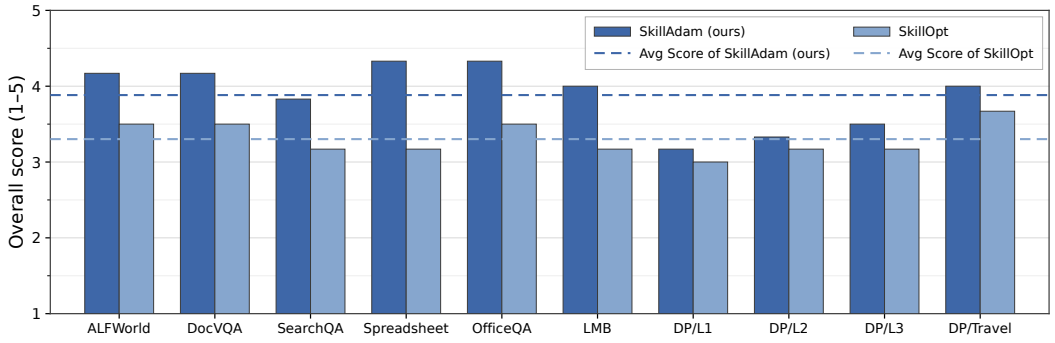


Figure 3: Overall skill-quality scores on a 1–5 scale across ten benchmark slices, evaluated by the same GPT-5.5 judge with method names hidden. Each value averages the six dimension medians from three independent calls. Horizontal markers show the mean across slices. Appendix D provides the evaluation protocol and detailed analysis.

Table 1: Main results on five short-horizon benchmarks. Baselines follow the GPT-5.5 no-harness results reported by SkillOpt. Scores are percentages. Bold and underlining mark the best and second-best values.

Method	SearchQA	Spreadsheet	OfficeQA	DocVQA	LiveMath
NoSkill	77.7	41.8	33.1	78.8	37.6
HumanSkill	81.8	72.9	66.9	90.1	38.4
LLMSkill	80.9	43.2	51.7	89.6	40.0
Trace2Skill	82.4	49.6	65.7	90.6	52.0
TextGrad	81.4	41.1	42.0	87.2	49.2
GEPA	84.8	73.6	63.9	89.1	43.2
SkillOpt	<u>87.3</u>	<u>80.7</u>	**72.1**	<u>91.2</u>	<u>66.9</u>
SKILLADAM	**87.5**	**81.1**	**72.1**	**92.3**	**67.7**

Table 2: Main results on ALFWorld with GPT-5.5 and DeepPlanning with Claude Sonnet 4.5. DP-Avg averages the unrounded DP-Shopping and DP-Travel accuracies. Scores are percentages. Bold and underlining mark the best and second-best values.

Method	ALFWorld	DP-Shopping	DP-Travel	DP-Avg
NoSkill	83.6	31.7	0.0	15.8
SkillOpt	<u>87.3</u>	<u>41.7</u>	<u>1.7</u>	<u>21.7</u>
SKILLADAM	**89.6**	**45.0**	**11.7**	**28.3**

4.4 ABLATION STUDIES

We conduct a cumulative ablation on DeepPlanning. −B replaces the adaptive budget with a fixed prompt-level budget of four hunks per iteration. −B, −M additionally removes the EIT and all optimizer conversation history. All variants follow the same training and evaluation protocol; Appendix B.3 details the prompt and state changes.

Table 3: Cumulative ablation on DeepPlanning. M: EIT and optimizer history; B: budget adaptation (−B uses a fixed four-hunk budget). Scores are percentages; bold marks column bests.

Configuration	DP-Shopping				DP-Travel	DP-Avg
	L1	L2	L3	All		
SKILLADAM	52.0	**36.0**	**50.0**	**45.0**	**11.7**	**28.3**
−B	48.0	**36.0**	40.0	41.7	1.7	21.7
−B, −M	**56.0**	20.0	30.0	36.7	1.7	19.2
NoSkill	40.0	24.0	30.0	31.7	0.0	15.8

In Table 3, the full SKILLADAM achieves the best overall result, increasing DP-Avg from 19.2% with a fixed four-hunk budget and no optimization memory to 28.3%. With Optimization Memory fixed, replacing the four-hunk budget with the adaptive budget adds 6.7 percentage points, with the largest gain on DP-Travel. This supports Update Adaptivity in long-horizon planning: reducing the next edit scope after inconsistent case-level effects can protect previously correct constraints, whereas consistent improvements permit broader updates.

With a fixed four-hunk budget, retaining the EIT and optimizer conversation history raises DP-Avg from 19.2% to 21.7%, with gains on Shopping L2 and L3 despite a decrease on L1. These gains on the more difficult levels suggest that cross-iteration feedback helps preserve useful corrections. This cumulative ablation does not isolate the effect of memory with budget adaptation enabled, the interaction between the two mechanisms, or the separate contributions of the EIT and optimizer conversation history.

4.5 CROSS-MODEL TRANSFER

We deploy skills optimized by SKILLADAM and SkillOpt using GPT-5.5 verbatim on GPT-5.4-mini, without further optimization or adaptation, on six benchmarks supported by both backbones. Table 4

Table 4: Cross-model transfer from GPT-5.5 to GPT-5.4-mini. Score denotes target-model task performance. Retention measures the percentage of source-model performance preserved after transfer. Both metrics are percentages. Bold marks the better result between SkillOpt and SKILLADAM.

Method	Metric	SearchQA	DocVQA	Spreadsheet	ALFWorld	LiveMath	OfficeQA	Average
NoSkill	Score	75.9	71.4	36.1	73.1	14.7	22.1	48.9
SkillOpt	Score	**80.8**	90.4	61.4	66.4	28.2	51.2	63.1
SkillOpt	Retention	**92.6%**	**99.1%**	76.1%	76.1%	42.2%	71.0%	76.2%
SKILLADAM	Score	78.9	**91.2**	**65.0**	**82.8**	**33.1**	**55.8**	**67.8**
SKILLADAM	Retention	90.2%	98.8%	**80.1%**	**92.4%**	**48.9%**	**77.4%**	**81.3%**

reports GPT-5.4-mini scores and $\text{Retention} = s_{\text{tgt}}/s_{\text{src}}$, where s_{src} and s_{tgt} denote GPT-5.5 and GPT-5.4-mini scores, respectively. No-Skill results on GPT-5.4-mini are taken from Yang et al. (2026) as reference performance.

Table 4 shows that SKILLADAM achieves higher target-model scores on five of the six benchmarks and higher retention ratios on four of the six benchmarks. Compared with SkillOpt, SKILLADAM improves the average target-model score from 63.1% to 67.8% and the average retention ratio from 76.2% to 81.3%. The advantage remains after excluding ALFWorld, indicating that the improvement is not driven by a single benchmark; SearchQA is the only target-score exception.

A plausible explanation for the stronger transfer is that Optimization Memory consolidates recurring problems and evaluated solution attempts across iterations, encouraging task-level procedures rather than source-model-specific wording. The comparison evaluates the complete framework, however, and does not isolate which component is responsible for the gain.

4.6 TRAINING COST

We compare optimization-phase API use and resulting performance for SKILLADAM and SkillOpt on the four DeepPlanning slices. Both use Claude Sonnet 4.5 and the same optimization cases. Counts cover optimization and iteration, excluding initial-skill generation, final test evaluation, and unrelated smoke tests. We report input/output tokens and API requests; token counts come from raw API counters and do not represent dollar cost.

Table 5: Optimization-phase API use and DeepPlanning test performance. API-use deltas are relative reductions. Performance deltas are absolute percentage-point differences.

Metric	SkillOpt	SKILLADAM	Delta
Input Tokens	222.1M	72.5M	−67.3%
Output Tokens	4.5M	1.5M	−66.8%
Total Tokens	226.6M	74.0M	−67.3%
API Requests	9,071	2,830	−68.8%
Resulting test performance (higher is better)			
DP-Shopping	41.7%	**45.0%**	+3.3 pp
DP-Travel	1.7%	**11.7%**	+10.0 pp
DP-Avg	21.7%	**28.3%**	+6.7 pp

In Table 5, SKILLADAM reduces total token consumption by 67.3% and API requests by 68.8% compared with SkillOpt, while improving DP-Avg from 21.7% to 28.3%. Tokens per request are similar and slightly higher for SKILLADAM, so the reduction reflects fewer optimization requests and modification attempts. This is consistent with Optimization Memory avoiding repeated rediscovery of failures and the Volatility-driven Edit Budget reducing broad, weakly supported revisions. Measured by DP-Avg per million optimization tokens, SKILLADAM is approximately four times as efficient as SkillOpt.

5 ANALYSES

We analyze SKILLADAM and SkillOpt on DeepPlanning Shopping Level 1. Both methods use Claude Sonnet 4.5 and start from the same initial skill, which has 40% test case accuracy. One

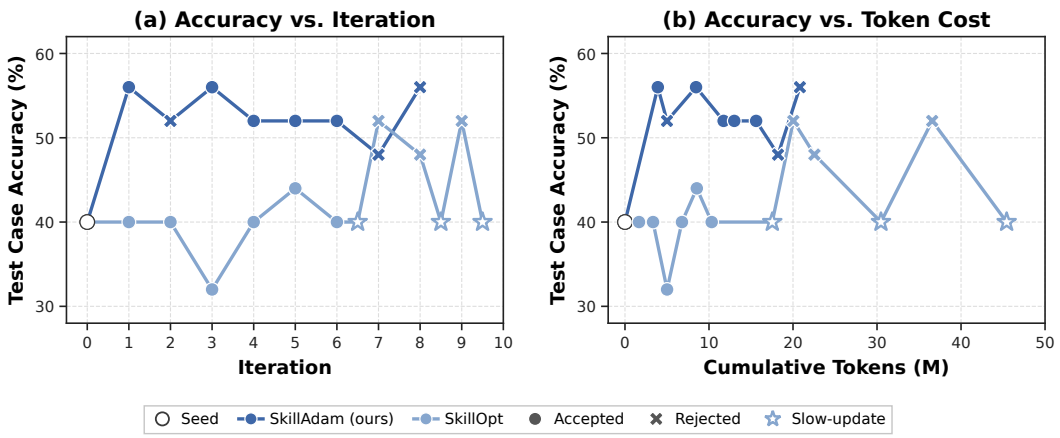


Figure 4: Optimization dynamics on DeepPlanning Shopping Level 1.

iteration denotes one minibatch-level modification attempt. SKILLADAM compares the current skill with its proposed revision on the same sampled optimization cases and applies its multi-metric acceptance gate. SkillOpt follows its native selection and slow-update rules. The markers in Figure 4 therefore record the decision made by each method's own update protocol.

The figure reports the post-hoc test accuracy of every evaluated modification. For the final retained skill, it reuses the test evaluation reported in the main results. Panel (a) uses the minibatch-level iteration index, and Panel (b) uses cumulative token consumption. Test performance is shown only for analysis and does not determine whether a modification is accepted. A modification with a high test score can still be rejected by the cases and metrics used in the corresponding update protocol.

5.1 STABILITY

In Figure 4(a), SKILLADAM finds a strong revision in the first iteration, and all subsequent accepted skills remain above the initial performance. In contrast, SkillOpt's accepted regular updates fluctuate around or below its starting performance, while its later high-scoring candidates are not retained. This comparison shows that SKILLADAM produces a more stable accepted optimization path.

The acceptance gate further prevents insufficiently supported revisions from replacing the current skill. Since the figure reports one run and memory operates together with the gate, it characterizes the overall optimization behavior rather than isolating the causal effect of memory alone.

5.2 EFFICIENCY

In Figure 4(b), SKILLADAM reaches an accepted skill with 56% test accuracy after approximately 4M tokens, whereas SkillOpt first produces a comparably strong candidate after approximately 20M tokens and does not accept it. SKILLADAM therefore reaches and retains a stronger skill with about one fifth of the token cost in this run.

Combined with the similar token cost per request in Table 5, the result shows that the efficiency gain comes from fewer unproductive modification attempts rather than cheaper individual calls.

6 CONCLUSION

We introduced **SKILLADAM**, a gradient-inspired framework for iterative skill self-evolution that adapts the two principles behind Adam to the skill space. An *optimization memory* accumulates structured records across iterations to provide stability, and a *volatility-driven edit budget* scales each update to the reliability of the improvement evidence to provide adaptivity. Across seven benchmarks that span short-horizon and long-horizon agentic tasks, SKILLADAM achieves state-of-the-art performance at substantially lower training cost. Its skills also transfer better across models.

AI USE STATEMENT

In this work, we used generative AI tools only for final-stage language polishing and grammar refinement of the manuscript. We have not used generative AI tools for generating research ideas, designing methods, conducting experiments, implementing code, analyzing data, interpreting results, or producing scientific claims, and other generative AI-assisted tasks requiring disclosure are not applicable to this work. We have reviewed all AI-assisted revisions to ensure their correctness and consistency with the authors' intended content. We take responsibility for the final content of this work, including all text, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We have made efforts to ensure the reproducibility of this work by providing detailed descriptions of the proposed method, experimental settings, implementation details, and evaluation protocols in the main paper and supplementary materials. The supplementary materials include the source code required to reproduce our experiments, and we will publicly release the code after the review process to further facilitate verification and future research. The content of this paper can be completely reproduced.

REFERENCES

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, A. Dimakis, Ion Stoica, Dan Klein, Matei A. Zaharia, and O. Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://arxiv.org/abs/2507.19457. Oral presentation.

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. URL https://arxiv.org/abs/2603.02766.

Anthropic. Introducing Agent Skills, October 2025. URL https://claude.com/blog/skills.

Minghao Chen, Yihang Li, Yanting Yang, Shiyu Yu, Binbin Lin, and Xiaofei He. AutoManual: Constructing instruction manuals by LLM agents via interactive environmental learning. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://papers.nips.cc/paper_files/paper/2024/hash/0142921fad7ef9192bd87229cdafa9d4-Abstract-Conference.html.

Chao Deng, Shaolei Zhang, Ju Fan, and Xiaoyong Du. DataEvolver: Automatic data preparation for large language models through multi-level self-evolving. *arXiv preprint arXiv:2606.07001*, 2026. URL https://arxiv.org/abs/2606.07001.

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. URL https://arxiv.org/abs/1704.05179.

Meihao Fan, Ju Fan, Yuxin Zhang, Shaolei Zhang, Xiaoyong Du, Jie Song, Peng Li, Fuxin Jiang, Tieying Zhang, and Jianjun Chen. DeepPrep: An LLM-powered agentic system for autonomous data preparation. *Proceedings of the VLDB Endowment*, 19(11):3371–3384, 2026. doi: 10.14778/3836663.3836695. URL https://www.vldb.org/pvldb/vol19/p3371-fan.pdf.

Srishti Gautam, Arjun Radhakrishna, and Sumit Gulwani. SkillAxe: Sharpening LLM-authored agent skills through evaluation-guided self-refinement. *arXiv preprint arXiv:2606.10546*, 2026. URL https://arxiv.org/abs/2606.10546.

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. *arXiv preprint arXiv:2604.01754v1*, 2026. URL https://arxiv.org/abs/2604.01754v1.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015. URL https://arxiv.org/abs/1412.6980.

Xiangyi Li, Yimin Liu, Wenbo Chen, Bingran You, Zonglin Di, Yifeng He, Shenghan Zheng, Kyoung Whan Choe, Jiankai Sun, Shuyi Wang, Chujun Tao, Binxu Li, Xuandong Zhao, Hejia Geng, Xiaojun Wu, Junwei Zhou, Xiaokun Chen, Hanwen Xing, Yubo Li, Qunhong Zeng, Di Wang, Yuanli Wang, Roey Ben Chaim, Penghao Jiang, Haotian Shen, Luyang Kong, Xinyi Liu, Runhui Wang, Xuanqing Liu, Jiachen Li, Xin Lan, Yueqian Lin, Wengao Ye, Junwei He, Songlin Li, Yue Zhang, Yipeng Gao, Yijiang Li, Ze Ma, Liqiang Jing, Tianyu Wang, Kaixin Li, Yiqi Xue, Haoran Lyu, Yizhuo He, Yuchen Tian, Shutong Wu, Bowei Wang, Yixuan Gao, Bo Chen, Litong Liu, Sikai Cheng, Jiajun Bao, Shuaicheng Tong, Shuwen Xu, Terry Yue Zhuo, Tinghan Ye, Qi Qi, Miao Li, Longtai Liao, Zelin Tan, Chang Shi, Xilin Tang, Srinath Tankasala, Boqin Yuan, Yaoyao Qian, Jianhong Tu, Chenguang Wang, Yizhou Sun, Wei Wang, Aaron Taylor, Ziyue Yang, Changkun Guan, Zhikang Dong, Xinyu Zhang, Steven Dillmann, Han chung Lee, and Dawn Song. SkillsBench: Benchmarking how well agent skills work across diverse tasks. *arXiv preprint arXiv:2602.12670v4*, 2026. URL https://arxiv.org/abs/2602.12670v4.

Yizhe Liu, Shaolei Zhang, and Ju Fan. DA-Studio: An agentic system for end-to-end data analysis. *Proceedings of the VLDB Endowment*, 19(12):4766–4769, 2026. doi: 10.14778/3827998.3828117. URL https://www.vldb.org/pvldb/vol19/p4766-liu.pdf.

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. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. URL https://arxiv.org/abs/2406.14991. Spotlight.

Minesh Mathew, Dimosthenis Karatzas, and C.V. Jawahar. DocVQA: A dataset for VQA on document images. In *IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 2021. URL https://arxiv.org/abs/2007.00398.

Jingwei Ni, Yihao Liu, Xinpeng Liu, Yutao Sun, Mengyu Zhou, Pengyu Cheng, Dexin Wang, Erchao Zhao, Xiaoxi Jiang, and Guanjun Jiang. Trace2Skill: Distill trajectory-local lessons into transferable agent skills. *arXiv preprint arXiv:2603.25158*, 2026. URL https://arxiv.org/abs/2603.25158.

Krista Opsahl-Ong, Arnav Singhvi, Jasmine Collins, Ivan Zhou, Cindy Wang, Ashutosh Baheti, Owen Oertell, Jacob Portes, Sam Havens, Erich Elsen, Michael Bendersky, Matei Zaharia, and Xing Chen. OfficeQA Pro: An enterprise benchmark for end-to-end grounded reasoning. *arXiv preprint arXiv:2603.08655*, 2026. URL https://arxiv.org/abs/2603.08655.

Siru Ouyang, Jun Yan, Yanfei Chen, Rujun Han, Zifeng Wang, Bhavana Dalvi Mishra, Rui Meng, Chun-Liang Li, Yizhu Jiao, Kaiwen Zha, Maohao Shen, Vishy Tirumalashetty, George Lee, Jiawei Han, Tomas Pfister, and Chen-Yu Lee. SkillOS: Learning skill curation for self-evolving agents. *arXiv preprint arXiv:2605.06614*, 2026. URL https://arxiv.org/abs/2605.06614.

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*, pp. 7957–7968. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.494. URL https://aclanthology.org/2023.emnlp-main.494/.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew J. Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations (ICLR)*, 2021. URL https://arxiv.org/abs/2010.03768.

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

Zora Zhiruo Wang, Apurva Gandhi, Graham Neubig, and Daniel Fried. Inducing programmatic skills for agentic tasks. In *Conference on Language Modeling*, 2025a. URL https://openreview.net/forum?id=lsAY6fWsog.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 63897–63911. PMLR, 2025b. URL https://proceedings.mlr.press/v267/wang25bx.html.

Cilin Yan, Jingyun Wang, Lin Zhang, Ruihui Zhao, Xiaopu Wu, Kai Xiong, Qingsong Liu, Guoliang Kang, and Yangyang Kang. Efficient and accurate prompt optimization: The benefit of memory in exemplar-guided reflection. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*, pp. 753–779. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.37. URL https://aclanthology.org/2025.acl-long.37/.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2309.03409.

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

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. URL https://arxiv.org/abs/2406.07496.

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

Shaolei Zhang, Ju Fan, Meihao Fan, Guoliang Li, and Xiaoyong Du. DeepAnalyze: Agentic large language models for autonomous data science. *arXiv preprint arXiv:2510.16872*, 2025. URL https://arxiv.org/abs/2510.16872.

Yinger Zhang, Shutong Jiang, Renhao Li, Jianhong Tu, Yang Su, Lianghao Deng, Xudong Guo, Chenxu Lv, and Junyang Lin. DeepPlanning: Benchmarking long-horizon agentic planning with verifiable constraints. *arXiv preprint arXiv:2601.18137*, 2026b. URL https://arxiv.org/abs/2601.18137.

Yuxin Zhang, Ju Fan, Meihao Fan, Shaolei Zhang, and Xiaoyong Du. CoDA-Bench: Can code agents handle data-intensive tasks? In *Proceedings of the 43rd International Conference on Machine Learning*, 2026c. URL https://arxiv.org/abs/2606.15300.

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

A Adaptive Moment Estimation

A.1 Adam Update

In differentiable optimization, Adam (Kingma & Ba, 2015) maintains exponential moving averages of the first and second moments of stochastic gradients. Let $\nabla_t = \nabla_\theta \mathcal{L}_t(\theta_{t-1})$ denote the stochastic gradient of the mini-batch loss at iteration t. Adam updates

$$m_t^{\mathrm{Adam}} = \beta_1 m_{t-1}^{\mathrm{Adam}} + (1-\beta_1)\nabla_t, \tag{8}$$

$$v_t^{\mathrm{Adam}} = \beta_2 v_{t-1}^{\mathrm{Adam}} + (1-\beta_2)\nabla_t^2, \tag{9}$$

Here, m_t^{Adam} and v_t^{Adam} are the first- and second-moment estimates. The coefficients $\beta_1, \beta_2 \in [0, 1)$ are exponential decay rates. The square is applied element-wise. Since both moment estimates are initialized at zero, Adam applies bias correction:

$$\widehat{m}_t^{\mathrm{Adam}} = \frac{m_t^{\mathrm{Adam}}}{1-\beta_1^t}, \qquad \widehat{v}_t^{\mathrm{Adam}} = \frac{v_t^{\mathrm{Adam}}}{1-\beta_2^t}. \tag{10}$$

The parameters are then updated as

$$\theta_t = \theta_{t-1} - \alpha \frac{\widehat{m}_t^{\mathrm{Adam}}}{\sqrt{\widehat{v}_t^{\mathrm{Adam}}} + \epsilon}, \tag{11}$$

where $\alpha > 0$ is the base learning rate and $\epsilon > 0$ is a small constant that prevents division by zero and improves numerical stability. The first-moment estimate aggregates gradient information across iterations to stabilize the update direction, while the second-moment estimate rescales the base learning rate according to the recent squared-gradient magnitude, yielding an adaptive effective step size. The ratio

$$\alpha_t^{\mathrm{eff}} = \frac{\alpha}{\sqrt{\widehat{v}_t^{\mathrm{Adam}}} + \epsilon} \tag{12}$$

can be interpreted as the element-wise effective step size.

A.2 Functional Correspondence

Figure 2 provides a block-level comparison between Adam and SkillAdam, while Table 6 makes the correspondence between their optimization quantities explicit. The analogy is functional, not numerical. SkillAdam does not compute gradients in the discrete skill space. It constructs language-space states that serve the same optimization roles.

Table 6: Core functional correspondence between Adam and SkillAdam. The correspondence is functional, with evaluation feedback and a language-space patch serving the roles of the loss and gradient.

Adam	SkillAdam	Functional role
Parameters θ_{t-1}	Current skill S_{t-1}	Represent the optimized state
Prediction o_t	Execution trajectories $\mathcal{T}_t$	Record the execution output
Loss $\mathcal{L}_t$	Evaluation feedback $\mathcal{F}_t^{\mathrm{roll}}$	Evaluate the current behavior
Gradient ∇_t	Skill patch g_t	Provide the local update signal
First moment m_t^{Adam}	Evolving Issue Tracker M_t	Stabilize the update direction
Second moment v_t^{Adam}	Historical volatility V_t	Accumulate update variability
Effective step size α_t^{eff}	Edit budget σ_{t+1}	Adapt the update magnitude
Apply $\Delta\theta_t$	Apply and gate g_t	Update the optimized state
Updated parameter θ_t	Accepted skill S_t	Carry the state forward

The rollout feedback $\mathcal{F}_t^{\mathrm{roll}}$ plays a loss-like role by evaluating the current behavior, while the generated patch g_t serves as the gradient-like local update signal in the skill space. The Evolving Issue Tracker M_t accumulates issue histories and solution outcomes across iterations, analogous to Adam's first moment. Similarly, V_t aggregates recent improvement variability and determines the next edit budget σ_{t+1}.

B BENCHMARKS AND EXPERIMENTAL DETAILS

B.1 BENCHMARKS

We evaluate SKILLADAM on seven benchmarks that cover knowledge work and interactive planning. Six benchmarks contribute one evaluation slice each. DeepPlanning contributes Shopping Levels 1–3 and Travel EN, giving ten slices in total. Each benchmark is evaluated in its corresponding task environment using the task-specific metrics described below.

We classify SearchQA, SpreadsheetBench, OfficeQA Full, DocVQA, and LiveMathematicianBench as *short-horizon* benchmarks, as they require fewer than ten tool calls per task on average in our setting. ALFWorld and DeepPlanning are *long-horizon* benchmarks, requiring at least ten tool calls per task on average.

SearchQA. SearchQA (Dunn et al., 2017) pairs Jeopardy! questions and answers with text snippets retrieved by a search engine. The snippets contain both relevant evidence and distracting or incomplete text, requiring the agent to identify useful information and recover the answer from noisy context. Although the snippets originate from the web, the released task is answerable from the provided context.

SpreadsheetBench. SpreadsheetBench (Ma et al., 2024) evaluates spreadsheet manipulation using natural-language requests collected from real Excel users. Tasks require cell-level or sheet-level modifications in spreadsheets with diverse layouts, including multiple tables and non-standard table structures. Its evaluation applies a solution to multiple spreadsheet test cases with different cell values, testing whether the solution generalizes beyond the original input file.

OfficeQA Full. OfficeQA Full (Opsahl-Ong et al., 2026, Appendix B) evaluates grounded question answering over historical U.S. Treasury Bulletins. The released Full benchmark contains 246 questions, comprising the 133-question Pro subset and 113 additional easier questions. Its tasks require locating relevant evidence in a heterogeneous document collection, interpreting textual and tabular information, and performing quantitative analysis. We use the Full-version data adopted by SkillOpt (Yang et al., 2026); "OfficeQA" elsewhere in this paper refers to this version.

DocVQA. DocVQA (Mathew et al., 2021) evaluates question answering on document images, where the answer is a text span in the supplied page. Its questions cover documents with running text, tables, forms, and other layout structures, requiring the agent to connect the question to both textual content and spatial organization.

LiveMathematicianBench. LiveMathematicianBench (LMB; LiveMath in the tables) (He et al., 2026) evaluates research-level mathematical reasoning through multiple-choice questions derived from recent arXiv theorems. The benchmark covers diverse logical forms, including implication, equivalence, existence, and bounds, and uses proof sketches to construct plausible distractors that probe understanding of the underlying arguments. Each item has five answer choices, and evaluation checks whether the selected choice matches the correct answer. We use the LMB data supplied with SkillOpt (Yang et al., 2026).

ALFWorld. ALFWorld (Shridhar et al., 2021) pairs text-based interactive environments with embodied household tasks from ALFRED. In the text setting used here, an agent receives a goal and textual observations, then issues commands to navigate, locate objects, and perform operations such as cleaning, heating, cooling, or placing them. Tasks require a sequence of state-dependent actions, and episode success is determined by whether the requested goal is completed.

DeepPlanning. DeepPlanning (Zhang et al., 2026b) evaluates long-horizon planning in tool-accessible shopping and travel environments. Given a user request, the agent must retrieve relevant information and produce a shopping cart or travel itinerary that satisfies local requirements together with global constraints such as total budget and temporal feasibility. We use Shopping Levels 1–3 and Travel EN, and report case accuracy: exact product matching for the complete shopping cart, or satisfaction of all commonsense and personalized checks for travel.

Metrics. We report the following primary metrics under our evaluation protocol. SearchQA, OfficeQA, and LMB use Exact Match. SpreadsheetBench uses hard task success: an instruction succeeds only if its solution passes all associated spreadsheet test cases. DocVQA uses ANLS-hard in our experiments. ALFWorld uses episode goal-completion rate, measuring the fraction of episodes in which the task goal is achieved. DeepPlanning reports Shopping Case Accuracy and Travel Case Accuracy, measuring the fractions of completely correct shopping carts and travel plans that satisfy all required checks, respectively. DP-Shopping pools the three shopping levels by case count, with 25, 25, and 10 test cases. DP-Travel contains 60 test cases. We define $\text{DP-Avg} = (\text{DP-Shopping} + \text{DP-Travel})/2$ using the unrounded domain-level accuracies. It is not an equal average of the four DeepPlanning slices. All main results are percentages.

B.2 Baselines

We compare SkillAdam with seven baselines. **NoSkill** runs the target agent without an injected skill. **HumanSkill** uses an expert-authored skill, and **LLMSkill** uses a skill written in one LLM call. The optimization baselines are **Trace2Skill** (Ni et al., 2026), **TextGrad** (Yuksekgonul et al., 2024), **GEPA** (Agrawal et al., 2026), and **SkillOpt** (Yang et al., 2026).

Results Taken from Prior Work. For SearchQA, SpreadsheetBench, OfficeQA, DocVQA, and LMB, all seven baseline results (NoSkill, HumanSkill, LLMSkill, Trace2Skill, TextGrad, GEPA, and SkillOpt) are taken directly from the GPT-5.5 no-harness results reported by SkillOpt (Yang et al., 2026); we do not rerun these baseline experiments. The ALFWorld NoSkill result is also taken from SkillOpt.

Reproduced Results. We reproduce SkillOpt on ALFWorld and both NoSkill and SkillOpt on DeepPlanning in the task environments used for SkillAdam. Within each benchmark, the compared methods use the same target-agent configuration, test cases, and evaluator. SkillAdam and SkillOpt share the same initial skill, constructed from fixed baseline execution trajectories.

SkillOpt retains its original train and selection split together with its native selection, slow-update, and optimizer-memory procedures. We run SkillOpt for four epochs under this protocol.

B.3 Experimental Setup

Common Protocol. All experiments use a no-harness, direct-chat setting. The agent interacts with the native task interface and evaluator of each benchmark without an additional orchestration layer. Skills are injected as natural-language instructions, and the target model remains frozen during optimization. We use GPT-5.5 for the six benchmarks outside DeepPlanning. These runs use medium reasoning effort, a temperature of 1.0, and a maximum output length of 16,384 tokens. DeepPlanning uses Claude Sonnet 4.5 with a temperature of 0.0 and the same output limit. Travel EN uses a separate frozen model to convert the generated plan into the required format before evaluation.

Initialization and Optimization. SkillAdam and SkillOpt share the same initial skill, constructed from fixed baseline execution trajectories. For the six benchmarks outside DeepPlanning, SkillAdam merges the original train and selection partitions into one optimization pool. SkillOpt retains the original split. The test partition is unchanged and is excluded from optimization; it is used for final-skill evaluation and the post-hoc analysis in Section 5. DeepPlanning uses an odd-even split by case identifier. Odd-numbered cases are used for optimization, and even-numbered cases are reserved for testing. We use seed 42 for local case sampling, optimization-pool shuffling, and other controlled random operations.

At each iteration, SkillAdam evaluates the current skill and its proposed revision on the same sampled cases. Their execution trajectories and evaluation feedback are passed to the optimizer states described in Section 3.2.

Edit Budget. Table 7 reports the budget configuration for each benchmark. Budgets are measured in unified-diff hunks. Per-case scores are on a $[0, 1]$ scale, and $V_{\max}$ is specified for the sample variance of candidate-minus-current score differences on that scale. The initial volatility is $V_0 = 0$,

giving $\sigma_1 = b_{\text{base}}$. The lower bound $b_{\min}$ constrains the requested budget, not the number of edits that must be generated; an empty patch remains valid.

Table 7: Benchmark-specific adaptive edit-budget settings. All benchmarks use the adaptive budget.

Benchmark	β_2	b_{base}	$b_{\min}$	$V_{\max}$	Volatility metric
SearchQA	0.9	4	1	0.05	Exact Match
SpreadsheetBench	0.9	4	2	0.10	Hard task success
OfficeQA	0.9	4	1	0.05	Exact Match
DocVQA	0.9	4	1	0.04	ANLS-hard
LiveMath/LMB	0.9	4	1	0.05	Exact Match
ALFWorld	0.9	4	1	0.10	Episode success
DP–Shopping L1–L3	0.9	4	1	0.05	`score`
DP–Travel EN	0.9	4	1	0.10	`composite_score`

Acceptance Protocol. Let Δm denote the candidate-minus-current difference in metric m, averaged over the same optimization batch. A candidate is accepted when at least one gain condition and all protection conditions in Table 8 hold. Thresholds are absolute differences on the $[0, 1]$ score scale. ALFWorld accepts non-decreasing success, including ties. DeepPlanning uses strict gain thresholds; the other benchmarks use inclusive gain thresholds. The negative regression boundaries in the table are strict. No separate validation set is used for these decisions. SkillOpt follows its native selection and slow-update rules.

Table 8: Benchmark-specific acceptance rules. Conditions in the gain column are joined by OR; conditions in the protection column are joined by AND.

Benchmark	Gain condition	Protection condition
SearchQA	$\Delta\text{EM} \geq 0.04$ or $\Delta\text{F1} \geq 0.025$	$\Delta\text{EM} > -0.04$ and $\Delta\text{F1} > -0.05$
SpreadsheetBench	$\Delta\text{hard} \geq 0.06$ or $\Delta\text{per-cell} \geq 0.03$	$\Delta\text{hard} > -0.08$ and $\Delta\text{per-cell} > -0.06$
OfficeQA	$\Delta\text{EM} \geq 0.08$ or $\Delta\text{F1} \geq 0.05$	$\Delta\text{EM} > -0.08$ and $\Delta\text{F1} > -0.05$
DocVQA	$\Delta\text{hard} \geq 0.01$ or $\Delta\text{soft} \geq 0.01$	$\Delta\text{hard} > -0.05$ and $\Delta\text{soft} > -0.05$
LiveMath/LMB	$\Delta\text{EM} \geq 0.08$	$\Delta\text{EM} > -0.08$
ALFWorld	$\Delta\text{hard} \geq 0$	Same-metric non-regression
DP–Shopping	$\Delta\text{case_score} > 0.15$ or $\Delta\text{match_rate} > 0.05$	$\Delta\text{case_score} > -0.15$ and $\Delta\text{match_rate} > -0.05$
DP–Travel	$\Delta\text{CS} > 0.03$ or $\Delta\text{PS} > 0.08$	$\Delta\text{CS} > -0.05$ and $\Delta\text{PS} > -0.08$

EM and F1 denote Exact Match and token F1. Per-cell denotes the spreadsheet cell pass rate. For DocVQA, soft is ANLS and hard indicates ANLS of at least 0.999. For Travel, CS and PS denote `commonsense_weighted_score` and `personalized_score`. LMB's protection condition is implied by its gain condition. Floating-point comparisons use an absolute tolerance of 10^{-12}.

Sampling and Stopping. Outside DeepPlanning, batches are drawn sequentially from a single traversal of the optimization pool; SpreadsheetBench first shuffles the pool with seed 42. A run ends when its available batches are exhausted, its iteration limit is reached, or its early-stopping condition holds. On DeepPlanning, each iteration samples five distinct cases from the optimization pool, and cases may recur across iterations. The iteration limits are 12 for each Shopping level and 20 for Travel. Early stopping requires at least one previously accepted update, completion of the minimum number of iterations in Table 9, and the specified number of consecutive iterations without an accepted update. Thus, DeepPlanning stops after four consecutive unaccepted iterations once at least five iterations have been completed for Shopping or seven for Travel.

Evaluation. For our final-skill evaluations, each skill is run once per test case with the target-agent configuration specified above. Figure 4 additionally evaluates intermediate candidate skills post hoc.

Table 9: Batch sizes and stopping settings. The minimum-iteration column controls when failure-based early stopping can activate; it does not prevent a sequential run from ending earlier if its batches are exhausted.

Benchmark	Batch	Max. iterations	Min. iterations	Unaccepted streak
SearchQA	15	60	40	10
SpreadsheetBench	15	15	15	99
OfficeQA	10	15	5	3
DocVQA	15	11	11	11
LiveMath/LMB	10	15	5	3
ALFWorld	10	6	6	99
DP–Shopping, each level	5	12	5	4
DP–Travel EN	5	20	7	4

For the final retained skill, the figure reuses the same test evaluation reported in the main results. These test scores are not used to select updates.

Ablation Configurations. In the $-\mathsf{B}$ variant, the requested budget is fixed at $\sigma_t = 4$ for every iteration, independently of the volatility estimate. The patch prompt asks for at most four unified-diff hunks; this remains a prompt-level constraint rather than a hard cap on realized edits. In the cumulative $-\mathsf{B}, -\mathsf{M}$ variant, we additionally disable EIT construction and updates, omit the tracker contents from the patch prompt, and remove optimizer conversation history across iterations. Patch generation therefore uses the current skill and current-batch trajectories and feedback without cross-iteration memory. All other training and evaluation settings are unchanged.

C Algorithm Details

C.1 Evaluation Feedback

Given a skill $S \in \mathcal{S}$ and a task $d \in \mathcal{D}$, the domain evaluator returns structured case-level evaluation feedback

$$\mathcal{E}(S, d) = (E(S, d), C(S, d)), \tag{13}$$

where $E(S, d) \in \mathbb{R}^m$ is an m-dimensional vector of domain-specific metrics, and $C(S, d)$ denotes optional diagnostic information, such as error descriptions or judge rationales. The metric vector E supports numerical aggregation and acceptance decisions, whereas C is retained in $\mathcal{F}$ as language-space evidence for patch generation and issue tracking. We use

$$\mathcal{F}(S, B) = \{\mathcal{E}(S, d) \mid d \in B\} \tag{14}$$

to denote the evaluation feedback collected over a task batch $B \subseteq \mathcal{D}$.

C.2 Expanded Framework Definitions

Framework Overview. Figure 2(b) presents the SkillAdam cycle. It begins with *rollout*, after which *moment estimation* informs the *skill update*. Before iterative optimization begins, SkillAdam constructs an initial skill S_0 from execution trajectories and their evaluation feedback. At iteration t, the current skill is executed on a sampled mini-batch, and the optimizer states carried from the previous iteration, M_{t-1} and σ_t, guide the generation of a candidate modification. After the candidate has been evaluated, the *Evolving Issue Tracker* compares the rollout and validation outcomes and evolves the structured issue list that constitutes the optimization memory M_t. The same paired feedback is used to update the improvement-volatility estimate.

The three blocks in Figure 2(b) describe functional roles. They do not prescribe a strict within-iteration execution order. In particular, the memory and volatility states obtained from the validation outcome of iteration t guide the skill update at iteration $t + 1$. The complete operational order is given in Algorithm 1, while Table 6 in Appendix A.2 summarizes the functional correspondence between Adam and SkillAdam.

Rollout. At iteration t, SKILLADAM samples a mini-batch $B_t \subset \mathcal{D}$ and executes the agent with the current skill S_{t-1}:

$$\mathcal{T}_t = \text{Rollout}\left(S_{t-1}, B_t\right), \tag{15}$$

where $\mathcal{T}_t$ denotes the collected execution trajectories. The domain evaluator then produces the corresponding case-level evaluation feedback

$$\mathcal{F}_t^{\text{roll}} = \mathcal{F}\left(S_{t-1}, B_t\right), \tag{16}$$

as defined in Equation 14. The trajectories record how the agent behaves during execution, while $\mathcal{F}_t^{\text{roll}}$ contains the resulting metric scores and diagnostic information. Together, $(\mathcal{T}_t, \mathcal{F}_t^{\text{roll}})$ provide the iteration-specific optimization signal used to generate the next skill update.

Moment Estimation. SKILLADAM maintains two persistent optimizer states across iterations. The optimization memory is instantiated as an *Evolving Issue Tracker* (EIT), denoted by M_t, while the volatility estimate V_t controls the magnitude of subsequent updates. Both states are refreshed after the candidate skill has been evaluated and are then carried into the next iteration.

The Evolving Issue Tracker is represented as a collection of structured issue entries:

$$M_t = \{I_j \mid j \in \mathcal{J}_t\}, \qquad I_j = (p_j, z_j, \mathcal{A}_j), \tag{17}$$

where $\mathcal{J}_t$ is the set of issue identifiers recorded through iteration t. The pair (p_j, z_j) describes an error pattern and its current status. The set $\mathcal{A}_j$ stores previous solution attempts with their observed outcomes. The tracker therefore maintains each issue and its resolution history.

After the candidate skill at iteration t has been evaluated, the tracker directly evolves its state:

$$M_t = \mathcal{U}_{\text{EIT}}\left(M_{t-1}, \mathcal{T}_t, \mathcal{F}_t^{\text{roll}}, g_t, \mathcal{F}_t^{\text{val}}, a_t\right). \tag{18}$$

The LLM-based update function $\mathcal{U}_{\text{EIT}}$ compares the rollout and validation outcomes to maintain the issue records. It links observed failures to existing issues when possible and creates entries when needed. It also records each attempted modification with its outcome and reopens an issue when the same failure recurs.

The Evolving Issue Tracker serves as a functional analogue of Adam's first-moment estimate:

$$m_t^{\text{Adam}} \quad \longleftrightarrow \quad M_t. \tag{19}$$

Both states integrate current evidence with information accumulated across previous iterations. Adam aggregates numerical gradients. The Evolving Issue Tracker instead accumulates issue histories and solution outcomes to stabilize the update direction.

SKILLADAM additionally estimates how consistently the candidate affects the evaluated cases. Let $\mathcal{E}_{t,i}^{\text{roll}}$ and $\mathcal{E}_{t,i}^{\text{val}}$ denote the case-level feedback for the current and candidate skills, respectively, on case $d_i \in B_t$. The case-level improvement is

$$\delta_{t,i} = s\big(\mathcal{E}_{t,i}^{\text{val}}\big) - s\big(\mathcal{E}_{t,i}^{\text{roll}}\big), \qquad d_i \in B_t, \tag{20}$$

where $s(\cdot)$ extracts the benchmark-specific scalar score used to measure improvement. The mean case-level improvement is

$$\bar{\delta}_t = \frac{1}{|B_t|} \sum_{d_i \in B_t} \delta_{t,i}. \tag{21}$$

SKILLADAM estimates the current improvement volatility using

$$\widehat{V}_t = \frac{1}{|B_t| - 1} \sum_{d_i \in B_t} \left(\delta_{t,i} - \bar{\delta}_t\right)^2, \qquad |B_t| \geq 2. \tag{22}$$

When fewer than two valid case-level comparisons are available, we set $\widehat{V}_t = 0$. A high value indicates that the candidate produces substantially different effects across cases, while a low value indicates more consistent effects.

The history-weighted volatility estimate is updated as

$$V_t = \beta_2 V_{t-1} + (1 - \beta_2)\widehat{V}_t, \qquad V_0 = 0. \tag{23}$$

The edit budget for the next iteration is then computed by

$$\sigma_{t+1} = \max\left(b_{\min}, \left\lfloor b_{\text{base}}\left[1 - \text{clip}\left(\frac{V_t}{V_{\max}}, 0, 1\right)\right]\right\rfloor\right), \tag{24}$$

where b_{base} is the base edit budget and $b_{\min}$ is the minimum allowable budget. The volatility saturation threshold is $V_{\max}$. High volatility yields a smaller edit budget, while low volatility permits a broader modification. The budget used at iteration t, σ_t, is determined from the state accumulated through iteration $t-1$.

This mechanism provides the following correspondence:

$$v_t^{\text{Adam}} \quad \longleftrightarrow \quad V_t, \qquad \alpha_t^{\text{eff}} \quad \longleftrightarrow \quad \sigma_{t+1}. \tag{25}$$

The optimization memory therefore determines which problems should guide the subsequent update, while the volatility-driven edit budget determines how extensively the skill may be modified.

Skill Update. At iteration t, the LLM-based patch generator proposes a skill modification conditioned on the current rollout evidence and the optimizer states carried from the previous iteration:

$$g_t = \mathcal{G}_{\text{LLM}}\left(S_{t-1}, \mathcal{T}_t, \mathcal{F}_t^{\text{roll}}, M_{t-1}, \sigma_t\right). \tag{26}$$

Here, $\mathcal{T}_t$ and $\mathcal{F}_t^{\text{roll}}$ describe the behavior observed in the current iteration, M_{t-1} provides cross-iteration guidance on which problems should be addressed, and σ_t controls the allowable scope of the modification.

Applying the generated modification to the current skill produces a candidate:

$$\widetilde{S}_t = \text{Apply}\left(S_{t-1}, g_t\right). \tag{27}$$

The candidate is evaluated on the same mini-batch used for the current-skill rollout:

$$\mathcal{F}_t^{\text{val}} = \mathcal{F}\left(\widetilde{S}_t, B_t\right). \tag{28}$$

Consequently, $\mathcal{F}_t^{\text{roll}}$ and $\mathcal{F}_t^{\text{val}}$ provide directly comparable case-level feedback for the current and candidate skills.

A benchmark-specific acceptance gate determines whether the candidate should replace the current skill:

$$a_t = G_\theta\left(S_{t-1}, \widetilde{S}_t, \mathcal{F}_t^{\text{roll}}, \mathcal{F}_t^{\text{val}}\right) \in \{0, 1\}. \tag{29}$$

The skill is updated as

$$S_t = \begin{cases} \widetilde{S}_t, & a_t = 1, \\ S_{t-1}, & a_t = 0. \end{cases} \tag{30}$$

The Evolving Issue Tracker uses each attempted modification and its evaluated outcome to evolve from M_{t-1} to M_t. The same paired feedback is used to compute $\widehat{V}_t$ and V_t. The updated skill and optimizer state are carried into the next iteration, closing the optimization loop shown in Figure 2(b).

C.3 Complete Algorithm

Algorithm 1 gives the operational order. States produced by evaluating a candidate at iteration t guide iteration $t+1$; a failed patch application leaves all states unchanged.

Algorithm 1 SKILLADAM: Adam-Inspired Skill Self-Evolution

Require: Task dataset $\mathcal{D}$, evaluator $\mathcal{E}$, mini-batch size k, maximum iterations $T_{\max}$, EMA coefficient β_2, budget parameters $b_{\text{base}}, b_{\min}, V_{\max}$, and random seed ξ
Ensure: Optimized skill $S_{T_{\max}}$

$(\mathcal{T}_0, \mathcal{F}_0) \leftarrow \text{CollectInitializationData}(\mathcal{D}, \mathcal{E}, \xi)$
$S_0 \leftarrow \text{Initialize}_{\text{LLM}}(\mathcal{T}_0, \mathcal{F}_0)$
$M_0 \leftarrow \emptyset$
$V_0 \leftarrow 0$
$\sigma_1 \leftarrow b_{\text{base}}$
for $t = 1, 2, \ldots, T_{\max}$ **do**
  $B_t \leftarrow \text{Sample}(\mathcal{D}, k, \xi + t), \quad \mathcal{T}_t \leftarrow \text{Rollout}(S_{t-1}, B_t), \quad \mathcal{F}_t^{\text{roll}} \leftarrow \mathcal{F}(S_{t-1}, B_t)$
  $g_t \leftarrow \mathcal{G}_{\text{LLM}}(S_{t-1}, \mathcal{T}_t, \mathcal{F}_t^{\text{roll}}, M_{t-1}, \sigma_t), \quad \widetilde{S}_t \leftarrow \text{Apply}(S_{t-1}, g_t)$
  if $\widetilde{S}_t = \textbf{null}$ **then**
    $S_t \leftarrow S_{t-1}, \quad M_t \leftarrow M_{t-1}, \quad V_t \leftarrow V_{t-1}, \quad \sigma_{t+1} \leftarrow \sigma_t$
    continue
  end if
  $\mathcal{F}_t^{\text{val}} \leftarrow \mathcal{F}(\widetilde{S}_t, B_t), \quad a_t \leftarrow G_\theta(S_{t-1}, \widetilde{S}_t, \mathcal{F}_t^{\text{roll}}, \mathcal{F}_t^{\text{val}})$
  if $a_t = 1$ **then**
    $S_t \leftarrow \widetilde{S}_t$
  else
    $S_t \leftarrow S_{t-1}$
  end if
  $M_t \leftarrow \mathcal{U}_{\text{EIT}}\left(M_{t-1}, \mathcal{T}_t, \mathcal{F}_t^{\text{roll}}, g_t, \mathcal{F}_t^{\text{val}}, a_t\right)$
  $\{\delta_{t,i}\}_{d_i \in B_t} \leftarrow \Delta\left(\mathcal{F}_t^{\text{roll}}, \mathcal{F}_t^{\text{val}}\right)$
  $\widehat{V}_t \leftarrow \text{Var}\left(\{\delta_{t,i}\}_{d_i \in B_t}\right)$
  $V_t \leftarrow \beta_2 V_{t-1} + (1 - \beta_2)\widehat{V}_t$
  $\sigma_{t+1} \leftarrow \max\left(b_{\min}, \left\lfloor b_{\text{base}}\left[1 - \text{clip}\left(\frac{V_t}{V_{\max}}, 0, 1\right)\right]\right\rfloor\right)$
end for
return $S_{T_{\max}}$

C.4 PROMPT AND CODE RESOURCES

The anonymous supplementary code includes the complete prompts for $\mathcal{G}_{\text{LLM}}$ and $\mathcal{U}_{\text{EIT}}$, their dynamic input construction, and benchmark-specific resources. The patch prompt combines the current skill, condensed rollout trajectories, optimization memory, and the current edit budget; its output contains the patch and its rationale. The EIT updater receives the retained skill, tracker state, condensed training trajectories, proposed patch, and acceptance record. Table 10 identifies the relevant files. All paths are relative to the extracted `experiments/` directory.

Table 10: Prompt resources in the anonymous supplementary code.

Resource	File location
Patch system template	`skillbench/core/prompts/iteration_system_prompt.md`
DeepPlanning patch system template	`skillbench/benchmarks/deepplanning/prompts/iteration_system_prompt_v2_r2_baseline.md`
EIT system template	`skillbench/core/prompts/momentum_update_system.md`
Runtime prompt assembly	`skillbench/methods/proposed_prompts.py`; `skillbench/methods/proposed.py`
Edit-budget instruction	`skillbench/core/edit_budget.py`
EIT tool definitions	`skillbench/core/momentum.py`
Prompt resource index	`skillbench/experiments/prompt_resources.json`

D SKILL QUALITY ANALYSIS

Beyond task performance, we evaluate the optimized skills with a frozen GPT-5.5 judge. The comparison covers ten benchmark slices, including six individual benchmarks and four DeepPlanning slices. Each skill is evaluated through three independent calls to the same judge model. The input does not reveal the method name. Each call assigns a score from 1 to 5 on six dimensions. Task Alignment and Non-Obvious Insight measure relevance. Constraint Handling and Decision Framework measure operational guidance. Cognitive Load and Inference Efficiency measure usability. For each dimension, we take the median of the three scores. The overall score for a skill is the mean of these six medians. We then average the overall scores across the ten benchmark slices.

As illustrated in Figure 3, SKILLADAM obtains a higher skill-quality score than SkillOpt on all ten benchmark slices, increasing the mean score from 3.30 to 3.88. Because every paired comparison favors SKILLADAM, the improvement is consistent across tasks rather than being driven by a small number of benchmarks.

This result complements the task-performance results in Section 4.3. The benchmark metrics measure whether the agent completes the task, whereas the judge evaluates whether the skill provides relevant, operational, and usable guidance. The agreement between the two evaluations suggests that SKILLADAM improves not only downstream execution but also the quality of the skill document itself. This improvement is consistent with Optimization Memory preserving useful constraints and the Volatility-driven Edit Budget limiting uncontrolled revisions. However, the judge experiment evaluates the complete framework, and Figure 3 aggregates all six dimensions; it therefore isolates neither an individual component nor a single quality dimension.