SKILLGRACE: GROUP-RELATIVE ADVANTAGE-GUIDED CONTRASTIVE EDITING OF AGENT SKILLS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) have evolved from text generators into agents that reason, use tools, and interact with environments, while skills improve their behavior through reusable instructions without updating model parameters. In this paper, we argue that existing skill optimizers do not fully exploit the information in task trajectories: each execution of a validation task provides only a single reward signal to guide patch selection, and their computational cost increases substantially as task execution becomes more complex. Moreover, aggregating evidence across diverse tasks can obscure the behaviors responsible for success or failure, making the proposal of effective candidate edits challenging and thereby highly dependent on strong external LLM editors. To address these limitations, we introduce SKILLGRACE, which formulates skill editing as group-relative policy optimization over an external language policy. To enable discrete skill optimization, SKILLGRACE samples multiple rollouts for each task and leverages group-relative advantages to construct same-task contrastive evidence for patch generation. The proposed patches are then selected by replaying these trajectories under candidate skills, eliminating the need for a separate validation set and substantially reducing optimization cost. Notably, since each skill patch is informed by contrastive evidence from same-task rollouts, SKILLGRACE enables the executor itself to generate effective edits without requiring a powerful external LLM editor. Across three benchmarks, SKILLGRACE outperforms SKILLOPT by 2.8 percentage points on average and by up to 6.8 points, while using 77.6% fewer optimization rollouts overall. An ALFWorld ablation further shows that same-task contrastive evidence improves success by 13.4 points over cross-task collections, highlighting its importance for effective self-evolution.

1 INTRODUCTION

Large language models (LLMs) can now reason, use tools, and act on feedback, enabling agents to solve tasks that extend far beyond single-turn generation (Yao et al., 2022; Schick et al., 2023). As these capabilities expand, agents are increasingly deployed on complex tasks involving long interaction histories, external tools and models, changing states, and delayed verification. Reliable performance on such tasks depends not only on model capabilities but also on reusable operational knowledge. Skills encode this knowledge as reusable instructions that guide agent behavior without updating model parameters (Anthropic, 2025b). This has motivated a growing line of work on skill optimization, which refines skills using agent trajectories and task feedback.

One line of work is reflection, which uses LLM-generated diagnoses of past experience to propose behavioral revisions (Shinn et al., 2023; Madaan et al., 2023; Yuksekgonul et al., 2024). Building on this paradigm, Trace2Skill analyzes execution traces and consolidates trajectory-level lessons into reusable skills (Ni et al., 2026). However, reflection alone does not measure whether a proposed skill edit actually improves task performance and therefore provides no direct signal for selecting among candidate edits. Validation-gated methods address this limitation by evaluating candidates on scored tasks: GEPA selects reflectively evolved candidates based on task-level performance (Agrawal et al., 2025), EvoSkill selects evolved agent programs through held-out evaluation (Alzubi et al., 2026), and SkillOpt accepts bounded skill edits only when they improve a held-out validation score (Yang et al., 2026a). Validation thus provides an empirical signal for retaining beneficial revisions and rejecting regressions.

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.

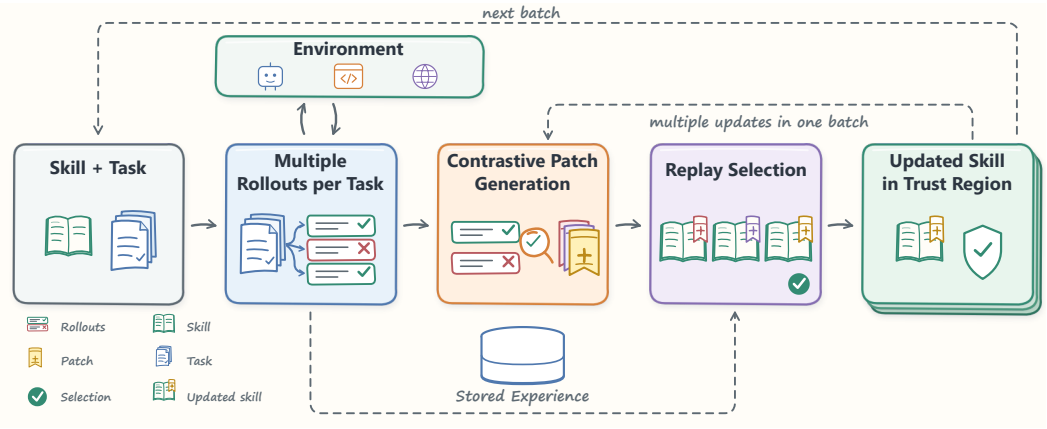

Figure 1: Overview of SKILLGRACE. For each task batch, agent is executed multiple times to collect same-task rollouts and their rewards through interacting with the environment. Group-relative advantages organize the corresponding trajectories into same-task contrastive evidence, which guides discrete skill patch generation. Candidate skills are then ranked by replaying the stored experience, as indicated by the bottom dashed path, without collecting fresh rollouts. Each selected patch updates the skill within a trust region. Patch generation, replay selection, and skill updating may repeat multiple times on the same stored experience. The resulting skill then follows the top dashed path to the next task batch.

Despite these advances, existing skill optimizers do not fully exploit the information contained in task trajectories. For candidate selection, each validation execution typically contributes only a single scalar reward, providing limited evidence about whether a skill edit genuinely improves performance. Because outcomes on complex tasks can vary across rollouts, reliable selection requires repeatedly executing every candidate, causing the optimization cost to grow rapidly with task complexity. Moreover, trajectory information is also underutilized in candidate generation: trajectories from diverse tasks are often aggregated and passed to an LLM editor, entangling task-specific differences with the behavioral choices responsible for success or failure. This obscures actionable evidence for skill revision and makes the generation of effective edits highly dependent on a powerful strong LLM editor. For example, SkillOpt evaluates executors ranging from Qwen3.5–4B to GPT-5.5, while using GPT-5.5 as the editor consistently in its main configuration (Yang et al., 2026a).

To address these limitations, we propose SKILLGRACE (Group-Relative Advantage-guide Contrastive Editing), which formulates skill editing as group-relative policy optimization over an external language policy. To enable optimization over the discrete space of textual skills, SKILLGRACE samples multiple rollouts for each task and uses group-relative advantages to organize the corresponding trajectories into *same-task contrastive evidence* for patch generation. This evidence reveals which behaviors should be reinforced or suppressed, enabling the executor itself to generate targeted skill patches without relying on a strong external LLM editor. The resulting patches are then selected by replaying the same trajectories under candidate skills. By grounding patch generation and selection in the same rollout batch, SKILLGRACE eliminates the need for a separate validation set and repeated validation interactions, substantially reducing optimization cost. Replay further enables multiple local updates without additional environment interaction, while a trust-region constraint limits each update's deviation from the current skill.

We evaluate SKILLGRACE on long-context answer extraction (SearchQA (Dunn et al., 2017)), tool-based spreadsheet manipulation (SpreadsheetBench (Ma et al., 2024)), and sequential environment interaction (ALFWorld (Shridhar et al., 2021)), using Qwen3.6–35B-A3B (Qwen Team, 2026) as the executor. We find that SKILLGRACE outperforms SKILLOPT, the strongest baseline (Yang et al., 2026a), on all three benchmarks, improving the average success rate by 2.8 percentage points while using 77.6% fewer optimization rollouts. Moreover, SKILLOPT uses GPT-5.5 as an external editor, whereas SKILLGRACE implement self-evolution by using Qwen3.6–35B-A3B as the editor. An

ALFWorld ablation further isolates the contribution of same-task contrastive evidence: it improves success by 13.4% over cross-task collections, supporting its role in effective self-evolution.

2 PROBLEM SETUP

We consider an agent system $\Phi = (M_\theta, \mathcal{H})$, where M_θ is a frozen language model and $\mathcal{H}$ is the agent harness that orchestrates the interaction process and mediates access to tools and the external environment. A skill ω, such as `skill.md`, is placed in the model context and remains fixed throughout an episode. The skill and frozen agent system jointly induce an external language policy:

$$\pi_\omega(a_t \mid s_t) := \Phi(a_t \mid s_t, \omega). \tag{1}$$

where s_t is the agent-visible state at interaction step t, containing the task input and interaction history, and a_t is the agent's next action, such as a language response or tool call.

Let $\mathbb{P}_{\text{task}}$ denote the task distribution over pairs (x, y), where x is a task input and y is its verification target, such as a ground-truth answer, evaluation rubric, or test suite. Executing Φ on x with ω produces a trajectory $\tau = (s_0, a_0, \ldots, s_T, a_T) \sim \pi_\omega(\cdot \mid x)$, and an evaluator R maps (τ, y) to a scalar reward. Skill optimization seeks to maximize expected reward over this task distribution:

$$\omega^\star \in \arg\max_\omega J(\omega), \qquad J(\omega) = \mathbb{E}_{(x,y)\sim\mathbb{P}_{\text{task}}} \mathbb{E}_{\tau\sim\pi_\omega(\cdot|x)}[R(\tau, y)]\,. \tag{2}$$

Interaction-efficient skill optimization. A rollout comprises the complete execution of Φ with ω and its evaluation by R. Such rollouts can require long generation, tool use, external interaction, and delayed verification. Given a budget of at most Q rollouts, the optimizer must make full use of each recorded trajectory because of the resource of agent execution. SKILLGRACE addresses this objective by reusing grouped training experience for patch generation, candidate selection, and multiple skill updates from a single batch.

3 METHOD

Given a finite training dataset $\mathcal{D}_{\text{train}} = \{(x_i, y_i)\}_{i=1}^N$ drawn from $\mathbb{P}_{\text{task}}$, SKILLGRACE alternates between online experience collection and offline replay. At iteration u, the current skill ω_u is used to collect multiple rollouts for each task. Their rewards provide group-relative advantages, and their trajectories provide contrastive evidence for an editor that proposes skill patches. Replay then selects the resulting candidate skills by ranking their likelihoods on the stored trajectories from the same batch, avoiding a separate validation set. Algorithm 1 summarizes the complete optimization cycle.

3.1 GROUP-RELATIVE EXPERIENCE

At the beginning of iteration u, We sample a batch $\{(x_i, y_i)\}_{i=1}^B$ from $\mathcal{D}_{\text{train}}$. For each task x_i in the batch, we sample K rollouts under ω_u, each of which yields a trajectory $\tau_{i,k}$ and reward $R_{i,k} = R(\tau_{i,k}, y_i)$, where $k \in \{1, \ldots, K\}$ denote the index of rollouts. We define the *group-relative advantage* of a rollout as its standardized reward relative to the other rollouts of the same task:

$$A_{i,k} = \text{clip}\left(\frac{R_{i,k} - \mu_i}{\sigma_i + \epsilon_A}, -A_{\max}, A_{\max}\right), \tag{3}$$

where $\mu_i = \frac{1}{K}\sum_{k=1}^K R_{i,k}$ and $\sigma_i = \sqrt{\frac{1}{K}\sum_{k=1}^K (R_{i,k} - \mu_i)^2}$ are the within-group reward mean and standard deviation, respectively; $\epsilon_A > 0$ is a small constant, and $A_{\max} > 0$ is the advantage clipping threshold, which limits the influence of extreme normalized advantages.

The sign of $A_{i,k}$ indicates whether a rollout performs above or below its same-task peers, while $|A_{i,k}|$ indicates how informative the corresponding trajectory is for distinguishing better and worse behavior. Together, the sign and magnitude of the advantage provide an essential signal for skill optimization. Moreover, since all rollouts in a group share the same task, within-group normalization eliminates cross-task variation in difficulty from the relative comparison. To make full use of these advantages in subsequent contrastive patch generation and replay selection, SKILLGRACE stores each advantage together with its corresponding task, target, trajectory, and reward as experience: $\mathcal{B}_u := \big\{(x_i, y_i, \tau_{i,k}, R_{i,k}, A_{i,k}) \mid 1 \le i \le B,\ 1 \le k \le K\big\}$.

Algorithm 1 Online skill optimization with SKILLGRACE.

Require: Inputs: frozen model M_θ, initial skill ω_0, training dataset $\mathcal{D}_{\text{train}}$, evaluator R
Require: Hyperparameters: iterations U, batch size B, group size K, replay limit E, patch subset size K_{patch}, replay subset size K_{replay}, gain margin η, KL budget δ

for $u = 0, \dots, U-1$ **do**
  // **Collect group-relative experience**
  $\{(x_i, y_i)\}_{i=1}^{B} \leftarrow \text{Sample}(\mathcal{D}_{\text{train}}, B)$
  $\mathcal{B}_u \leftarrow$ collect K rollouts per task under ω_u (Eq. equation 3)
  $\widetilde{\omega} \leftarrow \omega_u$
  for $e = 0, \dots, E-1$ **do**
    // **Generate patches from contrastive evidence**
    Patch anchors $\leftarrow$ global top-K_{patch} trajectories in $\mathcal{B}_u$ by $|A_{i,k}|$
    Patch evidence $\leftarrow$ each anchor and its same-task peers
    $\mathcal{P} \leftarrow$ editor converts the contrastive evidence into atomic patches for $\widetilde{\omega}$
    // **Select a patch by replay**
    $\mathcal{S}_{\text{replay}} \leftarrow$ global top-K_{replay} trajectories in $\mathcal{B}_u$ by $|A_{i,k}|$
    $\widehat{p} \leftarrow \arg\max_{p \in \mathcal{P}} \widehat{L}_{\text{replay}}(\widetilde{\omega} \oplus p; \mathcal{S}_{\text{replay}})$ (Eq. equation 4)
    $\omega_{\widehat{p}} \leftarrow \widetilde{\omega} \oplus \widehat{p}$
    // **Confirm the winner on the full experience**
    $\Delta_{\widehat{p}} \leftarrow \widehat{L}_{\text{replay}}(\omega_{\widehat{p}}; \mathcal{B}_u) - \widehat{L}_{\text{replay}}(\widetilde{\omega}; \mathcal{B}_u)$ (Eq. equation 4)
    $d_{\widehat{p}} \leftarrow \widehat{D}_{\text{KL}}(\pi_{\omega_u} \,\|\, \pi_{\omega_{\widehat{p}}})$ (Eq. equation 5)
    if $\Delta_{\widehat{p}} < \eta$ **or** $d_{\widehat{p}} > \delta$ **then**
      break
    end if
    $\widetilde{\omega} \leftarrow \omega_{\widehat{p}}$
  end for
  $\omega_{u+1} \leftarrow \widetilde{\omega}$
end for
return Optimized skill $\omega^\star \leftarrow \omega_U$

Explanation. Each iteration collects BK fresh rollouts. The K_{patch} trajectories with the largest absolute advantages serve as anchors for contrastive patch generation, whereas the K_{replay} trajectories with the largest absolute advantages form the priority subset for candidate screening. The winning patch is then confirmed against the current skill using its full-buffer replay gain (Eq. equation 4) and behavioral KL (Eq. equation 5). The same stored experience supports at most E local updates before new rollouts are collected.

3.2 CONTRASTIVE PATCH GENERATION

Prior trajectory-based skill optimizers represent a proposed revision as a *patch* that converts execution evidence into a localized change to the skill (Ni et al., 2026; Yang et al., 2026a). We follow this representation: a patch p performs one add, remove, or replace operation, and applying it to the current skill produces the candidate skill $\omega_p = \omega \oplus p$. Each atomic patch acts as a discrete local gradient for skill evolution, allowing successive accepted edits to evolve the skill gradually while the trust region limits their cumulative behavioral shift.

Rather than aggregating trajectories across different tasks, SKILLGRACE constructs *same-task contrastive evidence* for patch generation. Specifically, given the stored experience $\mathcal{B}_u$, SKILLGRACE globally selects the K_{patch} rollout trajectories with the largest advantage magnitudes $|A_{i,k}|$ as anchors. These anchors capture outcomes that deviate most strongly from their same-task peers, highlighting behaviors that may warrant reinforcement or suppression. For each anchor, the remaining trajectories from the same task are retrieved to form a contrastive evidence group. Because all trajectories within a group address the same task, differences in their outcomes are less confounded by task difficulty, allowing the executor to more directly identify the behavioral choices associated with success or failure. For example, if a positive-advantage anchor verifies a tool result while unsuccessful peers answer immediately, the executor may propose an instruction to verify the result before answering. Conversely, if a negative-advantage anchor repeatedly invokes an irrelevant tool while higher-advantage peers proceed directly, the executor may remove or suppress this behavior. The identified behavioral contrasts are then translated into atomic skill patches. In this way,

group-relative advantages serve not merely as scalar scores but as a mechanism for organizing full trajectories into actionable evidence for discrete skill optimization.

3.3 REPLAY SELECTION

The same batch used to construct contrastive evidence for patch generation is also reused to select the next discrete skill update during replay. We denote the current replay skill by $\widetilde{\omega}$ while the set of patches that editor propose under $\widetilde{\omega}$ is denoted by $\mathcal{P}$. After the editor proposes $\mathcal{P}$ under $\widetilde{\omega}$, SKILLGRACE must determine which patch should serve as the discrete gradient for the next evolution step. Specifically, given $\mathcal{P}$ and $\widetilde{\omega}$, SKILLGRACE first constructs a candidate skill $\omega_p = \widetilde{\omega} \oplus p$ for each $p \in \mathcal{P}$. Then, rather than evaluating these candidates through fresh rollouts on a separate validation set, it replays the stored experience $\mathcal{B}_u$ from the current iteration. During replay, each candidate skill scores the model-generated tokens in a stored trajectory while conditioning on the original preceding tokens, environment observations, and tool outputs. The resulting likelihood changes thus indicate how the candidate reinforces or suppresses the observed behaviors, without requiring additional agent–environment interaction. More precisely, let $\ell^{\omega}_{i,k,n}$ denote the log probability of the n-th generated token of $\tau_{i,k}$ under skill ω. Relative to the current skill $\widetilde{\omega}$, candidate ω_p induces the token-level likelihood ratio $\rho_{i,k,n}(\omega_p) = \exp(\ell^{\omega_p}_{i,k,n} - \ell^{\widetilde{\omega}}_{i,k,n})$. For any set of replayed trajectories $\mathcal{S}$, these ratios are aggregated using the following token-normalized objective:

$$\widehat{L}_{\text{replay}}(\omega_p; \mathcal{S}) = \frac{1}{|\mathcal{S}|} \sum_{(i,k)\in\mathcal{S}} \frac{1}{N_{i,k}} \sum_{n=1}^{N_{i,k}} \min\Big(\rho_{i,k,n}(\omega_p) A_{i,k}, \text{clip}(\rho_{i,k,n}(\omega_p), 1 \pm \varepsilon)\, A_{i,k}\Big), \quad (4)$$

where $N_{i,k}$ is the number of generated tokens from $\tau_{i,k}$ included in replay, and $\mathcal{S}$ is a priority subset used to screen all candidates or the complete buffer used to confirm the winning candidate.

Advantage-weighted replay The advantage $A_{i,k}$ is central to selection: it favors candidates that increase the likelihood of positive-advantage behavior and decrease that of negative-advantage behavior. To make selection efficient, SKILLGRACE first constructs a priority subset containing the K_{replay} trajectories with the largest $|A_{i,k}|$ in $\mathcal{B}_u$ and uses it to screen all candidate skills. It then replays only the highest-scoring candidate on the complete buffer and compares it against the unchanged skill $\tilde{\omega}$. The corresponding patch is accepted only if it improves the full-buffer replay objective while satisfying the trust-region constraint. This staged procedure avoids full-buffer replay for every candidate while ensuring that the selected update remains supported by the complete rollout batch. More importantly, by grounding both patch generation and selection in the same experience, replay avoids the train–validation mismatch and removes the need for fresh environment interaction during candidate evaluation.

3.4 MULTIPLE LOCAL UPDATES IN ONE BATCH

After accepting a patch, SKILLGRACE may generate and replay another set of patches using the updated skill and the stored experience from the same batch. This replay cycle permits up to E local updates without collecting fresh rollouts. The skill ω_u remains the reference throughout the cycle, making cumulative change measurable.

Repeated replay can eventually shift the skill beyond the evidence contained in the stored experience. We therefore measure the cumulative behavioral shift from the skill ω_u. Let $Z = \sum_{i,k} N_{i,k}$. For candidate ω_p, the sampled empirical KL is

$$\widehat{D}_{\text{KL}}(\pi_{\omega_u} \,\|\, \pi_{\omega_p}) = \frac{1}{Z} \sum_{i=1}^{B} \sum_{k=1}^{K} \sum_{n=1}^{N_{i,k}} \left(\ell^{\omega_u}_{i,k,n} - \ell^{\omega_p}_{i,k,n}\right). \quad (5)$$

This quantity estimates the forward KL on token contexts visited by the skill. We accept a candidate only when $\widehat{D}_{\text{KL}}(\pi_{\omega_u} \,\|\, \pi_{\omega_p}) \le \delta$. Because every replay step compares the cumulative candidate with the original skill ω_u, a sequence of individually small patches cannot accumulate an unbounded policy shift. The KL constraint therefore limits departure from behavior supported by the current batch and reduces regression on previously successful behavior.

Replay stops when no candidate improves over the current skill by the required margin, the KL budget is exhausted, or E updates have been reached. The resulting skill becomes ω_{u+1} and collects the next online batch. Consequently, each iteration uses exactly BK fresh rollouts; patch generation, candidate selection, and up to E local updates reuse this fixed batch without any further environment interaction.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Benchmarks. We evaluated three benchmarks from the suite adopted by Yang et al. (2026a), each representing a distinct source of agentic complexity. **SearchQA** tests answer extraction from long, noisy contexts of retrieved snippets (Dunn et al., 2017). **SpreadsheetBench** evaluates multi-round code generation for executable spreadsheet manipulation (Ma et al., 2024), whereas **ALFWorld** evaluates sequential decision-making through repeated actions and observations in a persistent environment (Shridhar et al., 2021). Together, these benchmarks cover long-context search, tool-mediated execution, and multi-step environment interaction. We report each benchmark's native exact-match or success rate as a percentage on held-out test tasks; test outcomes are not used to generate or select skill patches. More details about benchmarks are provided in Appendix A.

Baselines. We compare SKILLGRACE with four baselines: *No skill*, which uses the executor without additional instructions; *LLM skill*, a one-shot skill generated by GPT-5.5 (OpenAI, 2026); TRACE2SKILL, which derives trajectory-level patches with separate success and failure analysts and consolidates them using the executor itself (Ni et al., 2026); and SKILLOPT, which reflects over separate groups of successful and failed trajectories, uses GPT-5.5 as an external editor, and selects candidate skills through held-out validation (Yang et al., 2026a). Together, these baselines cover unassisted execution, one-shot skill generation, self-evolution, and validation-gated optimization with a stronger editor, providing a broad basis for evaluating SKILLGRACE.

Models. We use Qwen3.6–35B-A3B (Qwen Team, 2026) as the frozen executor on all three benchmarks and update only the external skill. Both SKILLGRACE and TRACE2SKILL use self-evolution, with the Qwen executor also serving as the skill editor. In contrast, SKILLOPT uses GPT-5.5 as its editor by default. All methods execute tasks with the same base model, while their skill optimization procedures and editor models differ.

Harness. We use a lightweight benchmark-specific harness rather than a general-purpose agent runtime such as Codex (OpenAI, 2025) or Claude Code (Anthropic, 2025a). Qwen3.6–35B-A3B is deployed locally with vLLM (Kwon et al., 2023) and accessed through its OpenAI-compatible API. The harness injects the current skill once into the system message, while benchmark adapters construct task messages, mediate tool or environment interactions, record complete trajectories, and compute rewards. Training uses 30 iterations with four tasks and eight rollouts per task, followed by at most three replay updates on each stored batch. Only visible action tokens contribute to the replay objective; generated reasoning and environment observations remain context rather than scoring targets. Full hyperparameters and the task-specific protocols are provided in Appendix B.

4.2 MAIN RESULTS

Table 1 reports the results on three benchmarks. SKILLGRACE achieves the highest score on every benchmark and the highest average score of 73.7. Relative to the no-skill average of 58.3, it improves SearchQA by 6.3 points, SpreadsheetBench by 17.5 points, and ALFWorld by 22.4 points, yielding an average gain of 15.4 points. The largest gains occur on SpreadsheetBench and ALFWorld, where the skill must guide tool-mediated execution or a sequence of environment-dependent decisions rather than only produce a final answer.

SKILLOPT is the strongest baseline in every column, yet SKILLGRACE exceeds it by 0.2 points on SearchQA, 6.8 points on SpreadsheetBench, and 1.5 points on ALFWorld. This raises the average from 70.9 to 73.7, a gain of 2.8 points. Notably, SKILLGRACE obtains these gains using Qwen3.6–35B-A3B as both executor and patch editor, whereas SKILLOPT uses GPT-5.5 as a separate edi-

Table 1: Qwen3.6–35B-A3B performance (%) in our experiments. Avg. is the unweighted mean across the three benchmarks. Subscripts show percentage-point differences relative to No skill. "Self" indicates that the executor also serves as the editor.

Executor	Skill source	Editor	SearchQA	Spreadsheet	ALFWorld	Avg.
Qwen3.6–35B-A3B	No skill	—	73.7	31.1	70.1	58.3
	LLM skill	GPT-5.5	$76.6_{+2.9}$	$33.2_{+2.1}$	$76.8_{+6.7}$	$62.2_{+3.9}$
	Trace2Skill	Self	$78.5_{+4.8}$	$39.2_{+8.1}$	$79.1_{+9.0}$	$65.6_{+7.3}$
	SkillOpt	GPT-5.5	$79.8_{+6.1}$	$41.8_{+10.7}$	$91.0_{+20.9}$	$70.9_{+12.6}$
	SkillGrace	Self	$\mathbf{80.0}_{+6.3}$	$\mathbf{48.6}_{+17.5}$	$\mathbf{92.5}_{+22.4}$	$\mathbf{73.7}_{+15.4}$

Table 2: Number of rollouts used for skill optimization. Counts include task executions for generating and selecting skill updates and exclude final test evaluation.

Method	SearchQA	Spreadsheet	ALFWorld	Total
SKILLOPT	10,520	920	1,396	12,836
SKILLGRACE	960	960	960	2,880

tor. Moreover, the self-evolving TRACE2SKILL baseline and the one-shot LLM skill remain below SKILLGRACE on every benchmark. These results show that SKILLGRACE can produce effective updates without a stronger external editor. Within this framework, same-task contrastive evidence helps the executor identify behaviors to reinforce or suppress, supporting self-evolution that outperforms SKILLOPT despite its stronger editor. This finding is consistent with our motivation that better-organized trajectory evidence supports more effective skill evolution. The contribution of same-task contrast is examined separately in Section 4.4.

4.3 OPTIMIZATION EFFICIENCY

We measure interaction efficiency by the number of rollouts collected during skill optimization, where each rollout is one complete task execution. SKILLGRACE interacts with the environment only once to collect the rollouts of each batch; patch generation and selection rely on the stored experience without further environment interaction. In contrast, a validation-gated method must execute every proposed candidate skill on every task in the validation set. This Cartesian product makes selection increasingly expensive as either the candidate pool or validation set grows.

Table 2 instantiates these counts using SKILLOPT's default optimizer hyperparameters and our configuration. Across the three benchmarks, SKILLGRACE uses 2,880 rollouts, compared with 12,836 for SKILLOPT, a reduction of 77.6%. The reduction is 90.9% on SearchQA and 31.2% on ALFWorld. On Spreadsheet, SKILLGRACE uses 4.3% more rollouts (960 versus 920) but improves the benchmark score by 6.8 percentage points. Taken together with Table 1, SKILLGRACE raises the average score from 70.9 to 73.7 while using 77.6% fewer rollouts overall. This reduction is consequential for agent tasks because a rollout can require long-context generation in SearchQA, repeated program execution and repair in SpreadsheetBench, or as many as 50 stateful action–observation exchanges in ALFWorld. Replay amortizes these costly interactions by selecting candidates and performing multiple local updates through teacher-forced inference over stored experience, without generating new trajectories or interacting with the environment again.

4.4 EFFECT OF SAME-TASK CONTRASTIVE EVIDENCE

To evaluate whether same-task contrast provides more effective evidence for patch generation, we conduct an ablation on ALFWorld under two evidence-construction conditions. All experimental settings are held fixed, while the conditions differ only in how evidence is provided to the editor. *Same-task contrast*, our default condition, groups multiple rollouts of the same task. By contrasting trajectories with positive and negative group-relative advantages from the same task, the editor can diagnose which behavioral choices should be strengthened or suppressed without being confounded by task difficulty or context. Moreover, generating patches is even easier for the editor than solving

Table 3: ALFWorld ablation of the evidence used for patch generation. The cross-task condition uses successful and failed trajectories in separate cross-task groups. Δ is measured relative to same-task contrast.

Editor evidence	Success rate (%)	Δ
Cross-task evidence	79.1	-13.4
Same-task contrast	**92.5**	—

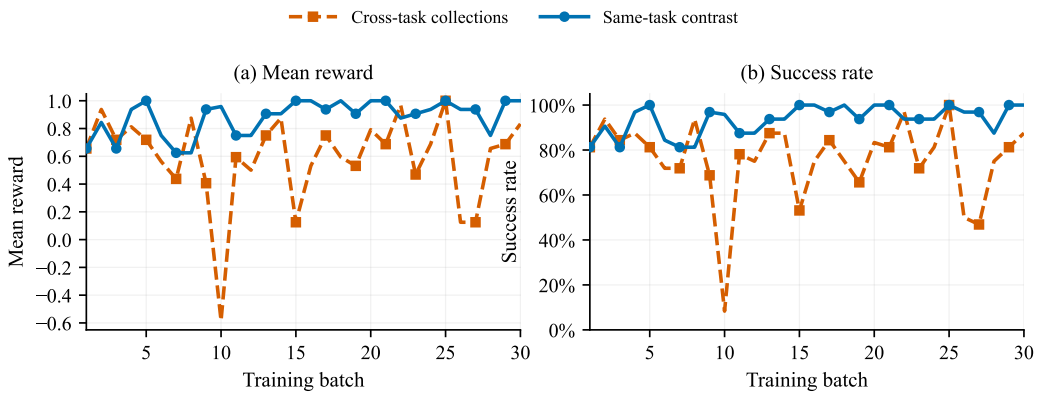

Figure 2: ALFWorld optimization dynamics with cross-task collections and same-task contrast. Curves report the raw means for each training batch without smoothing. Mean reward is the training reward, which assigns 1 to success, 0 to an ordinary unsuccessful rollout, and -1 to a terminal generation failure. Success rate is the fraction of completed episodes.

the task is for the executor, since the editor sees both the task and multiple trajectories with their outcomes. *Cross-task collections*, on the contrary, collects successful and failed trajectories from different tasks separately. Such evidence may still reveal broadly useful patterns, but the editor must first disentangle behavioral causes from differences in goals, states, and feasible actions across tasks. Patch generation therefore becomes more demanding and depends more strongly on the editor's reasoning capability. In this ablation, the training reward is 1 for success, 0 for an ordinary unsuccessful rollout, and -1 for token budget exhaustion caused by repeated generation. Appendix B provides the complete reward and evaluation protocol, while Appendix C reports the editor prompts used in both conditions.

As shown in Table 3, same-task contrast reaches a 92.5% evaluation success rate, compared with 79.1% for cross-task collections, a gain of 13.4 percentage points. The optimization curves in Figure 2 show that this benefit is not confined to the final skill. Across 30 training batches, same-task contrast attains an average success rate of 93.5%, compared with 76.0% for cross-task collections, and raises the mean training reward from 0.594 to 0.883. It is higher in 24 batches and tied in two under both measures, and ends with a 100.0% success rate versus 87.5% for cross-task collections. The largest difference occurs at batch 10, where same-task contrast achieves a 95.8% success rate compared with 8.3% for cross-task collections, a gap of 87.5 percentage points. The corresponding mean training rewards are 0.958 and -0.583. Such negative reward are caused by 14 rollouts exhausting the token budget, while all the rollouts in same-task contrastive avoid such behaviors successfully. Given the same setting of reward, the same-task contrastive evidence make the editor successfully capture such signals and suppress repeated generation through applying patches.

The two conditions use the same editor, trajectory budget, and optimization pipeline, the improvement comes directly from how the evidence is constructed. Cross-task collections still improves by 9.0 points over the no-skill executor, showing that the editor can extract useful general patterns from heterogeneous trajectories. However, its 13.4-point deficit relative to same-task contrast indicates that mixing task contexts places a substantially greater synthesis burden on the editor. Same-task

contrast instead turns patch generation into a focused comparison of behaviors under a fixed task, making useful update directions easier to identify. This benefit is particularly important for self-evolution, where the executor also serves as the editor, and supports same-task contrastive evidence as an essential contributor to the ALFWorld gains of SKILLGRACE.

5 RELATED WORK

Automatic prompt optimization. Automatic prompt optimization improves model behavior by revising natural language instructions rather than model weights. Reflection provides a natural starting point for language-based revision: Self-Refine uses self-feedback to revise outputs, Reflexion stores verbal feedback to guide subsequent agent trials, and TextGrad uses textual feedback to update components of LLM systems (Madaan et al., 2023; Shinn et al., 2023; Yuksekgonul et al., 2024). Since reflective feedback alone does not establish whether a revised instruction improves overall task performance, prompt optimizers therefore use evaluation scores to select candidates. APE selects LLM-generated instructions using task scores, while OPRO uses previous candidates and their scores to propose further revisions (Zhou et al., 2022; Yang et al., 2023). MIPRO extends performance-guided search to instructions and demonstrations in multi-stage LM programs (Opsahl-Ong et al., 2024). GEPA combines trace-based reflection to propose prompt updates with evaluation on a validation set to select them (Agrawal et al., 2025). Evaluation on the validation set provides a useful performance signal, but repeatedly validating candidates becomes costly especially when each validation requires a long agent interaction.

Agent skill optimization. Unlike a prompt for a simple chat, a skill is a reusable artifact that guides an agent to execute comlicated tasks. Optimizing such skill automatically grows a line of work. Trace2Skill consolidates lessons from execution trajectories into transferable skills (Ni et al., 2026), while EvoSkill discovers and refines structured skill folders through failure analysis and held-out validation (Alzubi et al., 2026). More directly, SkillOpt proposes bounded edits to a skill document and accepts candidates that improve a validation score (Yang et al., 2026a). These methods show that skills can be improved from experience. SKILLGRACE instead constructs patch evidence by contrasting rollouts of the same task, then selects candidates by replaying those rollouts without executing each candidate on a separate validation set.

Skill-augmented agents and agent self-improvement. A broader line of work uses skills to improve the agent as a whole. Voyager accumulates an executable skill library during exploration (Wang et al., 2023); AutoSkill derives and retrieves skills across interactions (Yang et al., 2026b); and SkillRL co-evolves a hierarchical skill library with the agent policy (Xia et al., 2026). Skill-Pro is particularly relevant because it learns procedural skills from experience and uses a non-parametric PPO-style gate to verify candidates before adding them to a maintained skill pool (Mi et al., 2026). Their objective is to acquire, select, and maintain reusable procedures for agent-level adaptation. By contrast, SKILLGRACE optimizes the given skill instead of a dynamic skill pool of agent as an external language policy while holding the model and agent harness fixed.

6 CONCLUSION

We introduced SKILLGRACE, which optimizes agent skills as external language policies without updating model weights. Group-relative advantages from same-task rollouts guide contrastive patch generation, while replay selects patches and supports multiple trust-region updates from each interaction batch. Across SearchQA, SpreadsheetBench, and ALFWorld, SKILLGRACE consistently outperforms the baselines while requiring substantially fewer optimization rollouts, even when the executor itself serves as the editor. The ALFWorld ablation further highlights the value of same-task contrastive evidence: it enables more effective skill edits than cross-task evidence. Together, these findings show that making fuller use of each collected trajectory can improve both skill quality and interaction efficiency.

AI USE STATEMENT

Large language models and coding agents are used to aid with and polish writing and generate some tables and plots. Additionally, we used coding agents for code generation to implement our proposal. We have reviewed all AI-assisted work. LLM-generated code was verified and tested for correctness. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Lakshya A. Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J. Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. *arXiv preprint arXiv:2507.19457*, 2025.

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.

Anthropic. Claude 3.7 Sonnet and Claude Code, February 2025a. URL https://www.anthropic.com/news/claude-3-7-sonnet.

Anthropic. Equipping agents for the real world with agent skills. https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills, 2025b. Accessed 2026-07-21.

Matthew Dunn, Levent Sagun, Mike Higgins, V. Ugur Güney, Volkan Cirik, and Kyunghyun Cho. SearchQA: A new q&a dataset augmented with context from a search engine. *CoRR*, abs/1704.05179, 2017. URL https://arxiv.org/abs/1704.05179.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626, 2023. doi: 10.1145/3600006.3613165.

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*, volume 37, 2024. doi: 10.52202/079017-3007. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/ac840df270ac537dd74530a15c332684-Abstract-Datasets_and_Benchmarks_Track.html.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*, 2023.

Qirui Mi, Zhijian Ma, Mengyue Yang, Haoxuan Li, Yisen Wang, Haifeng Zhang, and Jun Wang. Skill-Pro: Learning reusable skills from experience via non-parametric PPO for LLM agents. *arXiv preprint arXiv:2602.01869*, 2026.

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.

OpenAI. Codex CLI, 2025. URL https://github.com/openai/codex. Official repository; accessed 2026-09-25.

OpenAI. GPT-5.5 system card, April 2026. URL https://deploymentsafety.openai.com/gpt-5-5/gpt-5-5.pdf.

Krista Opsahl-Ong, Michael J. Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. Optimizing instructions and demonstrations for multi-stage language model programs. *arXiv preprint arXiv:2406.11695*, 2024.

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.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*, 2023.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*, 2023.

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

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.

Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, Zeyu Zheng, Cihang Xie, and Huaxiu Yao. SkillRL: Evolving agents via recursive skill-augmented reinforcement learning. *arXiv preprint arXiv:2602.08234*, 2026.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. *arXiv preprint arXiv:2309.03409*, 2023.

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*, 2026a.

Yutao Yang, Junsong Li, Qianjun Pan, Bihao Zhan, Yuxuan Cai, Lin Du, Jie Zhou, Kai Chen, Qin Chen, Xin Li, Bo Zhang, and Liang He. AutoSkill: Experience-driven lifelong learning via skill self-evolution. *arXiv preprint arXiv:2603.01145*, 2026b.

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.

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.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. *arXiv preprint arXiv:2211.01910*, 2022.

Table 4: Exact benchmark partitions adopted from the released SKILLOPT split manifests. Counts are numbers of task instances.

Benchmark	Train	Validation	Test
SearchQA	400	200	1,400
SpreadsheetBench	80	40	280
ALFWorld	39	18	134

A BENCHMARK DETAILS

We follow the fixed split manifests released in the official SKILLOPT repository (Yang et al., 2026a). Each manifest specifies the exact examples in the `train`, `val`, and `test` directories, so we use the released membership directly rather than generating a new random split. Table 4 summarizes the resulting partitions. During skill optimization, SKILLGRACE draws tasks only from the training split. We retain the validation split to match the released data protocol but do not use it to propose or select patches. The test split remains held out until final evaluation.

SearchQA. SearchQA pairs questions with noisy context assembled from retrieved Web snippets and one or more acceptable answers (Dunn et al., 2017). The agent receives the question and context in a single rollout and must return a grounded short answer. We report exact-match accuracy, which assigns a binary success to an answer that matches any reference answer after the benchmark's standard normalization. We use SKILLOPT's released `searchqa_id_split` manifest, whose IDs match the `key` field of the `lucadiliello/searchqa` release. Materializing these IDs yields 400 training, 200 validation, and 1,400 test examples.

SpreadsheetBench. SpreadsheetBench evaluates whether an agent can translate a natural-language instruction into executable spreadsheet manipulations on realistic workbook files (Ma et al., 2024). We use the expert-verified 400-instance subset distributed as `spreadsheetbench_verified_400`. Following the released SKILLOPT configuration, the agent operates in multi-round code-generation mode for at most three turns. The generated program is executed on the input workbook, and the resulting workbook is compared with the reference workbook at the designated answer cells; an instance succeeds only when the required cells match. We use the released `spreadsheetbench_id_split` membership, containing 80 training, 40 validation, and 280 test tasks.

ALFWorld. ALFWorld is an interactive text environment in which an agent must complete household goals through sequences of observations and admissible actions, such as navigating, manipulating objects, and placing them in specified receptacles (Shridhar et al., 2021). Each episode is scored by binary task completion. We use the `json_2.1.1` game files and the released `alfworld_path_split` manifest, which stores each game's path and task type. Its entries are taken from the repository's `split_train.json`, `split_val.json`, and `split_test.json` files and yield 39 training, 18 validation, and 134 test episodes. Following the released configuration, each rollout is limited to 50 environment steps.

B HARNESS AND IMPLEMENTATION DETAILS

Direct vLLM protocol. We implement a lightweight adapter for each benchmark rather than running the executor inside a general-purpose agent harness such as Codex (OpenAI, 2025) or Claude Code (Anthropic, 2025a). Qwen3.6–35B-A3B is served locally with vLLM 0.25.0 through an OpenAI-compatible API (Qwen Team, 2026; Kwon et al., 2023). For every rollout, the adapter places the current skill in the system message exactly once and presents the task through user messages. In interactive tasks, it then appends tool or environment observations to the same conversation and requests the next model action. Reference answers and evaluator diagnostics remain on the evaluator side and are never exposed to the executor. The adapter records the resulting conversation as a trajectory and computes the reward used by SKILLGRACE. Replay retains the recorded context but

never re-executes the task. Only visible action tokens contribute to the replay objective, as specified below. Held-out evaluation uses one rollout per test task.

Task and rollout sampling. Training tasks are sampled without replacement in shuffled cycles. A dedicated random-number stream, initialized with seed 42, shuffles the training indices once per cycle and divides them into consecutive batches of at most four tasks. Every task is visited before a new cycle begins; an incomplete final batch is kept rather than padded with repeated tasks. For each selected task, eight rollouts start from independent executions under the same fixed skill. Rollout seeds are assigned before dispatch to eight workers, so completion order does not change task membership or seed assignment. The task-order stream is separate from rollout sampling and editor calls. Consequently, changes in the number of proposed patches do not alter the task schedule. Actual rollout counts are the sum of the collected group sizes, including any smaller end-of-cycle batch, rather than necessarily the maximum batch size times the number of iterations.

The project's executor sampling configuration uses temperature 1.0, top-$p = 1.0$, disabled top-k and min-p filters, presence penalty 0, and repetition penalty 1.0. Thus, repeated rollouts sample the unmodified model distribution, whereas the editor uses the non-neutral generation settings in Table 5. Each adapter requests one completion per turn; eight rollouts denote eight complete executions, not eight tokens or eight continuation turns of one episode.

Visible action objective. We set `objective_token_scope=visible_action`. An exact token mask selects the public assistant content exposed to the benchmark: the answer for SearchQA, the generated Python program for SpreadsheetBench, and the action response for ALFWorld. Parser-separated reasoning, chat-template boundaries, task inputs, environment observations, and tool outputs are not objective tokens. They remain in the recorded prefix where applicable, so visible tokens are scored in their original context rather than with reasoning removed. The clipped replay objective and its KL, effective-sample-size, and clipping diagnostics all use this same mask. Each trajectory first averages over its selected tokens, and the objective then averages across scoreable trajectories. A trajectory with no visible action tokens remains in reward normalization, the experience archive, and editor evidence, but is excluded from replay scoring. Missing or ambiguous token alignment is not replaced by a guessed mask. Bounded reasoning excerpts may be supplied to the editor when the adapter allows them; this does not make reasoning tokens part of the objective.

Optimization hyperparameters. Table 5 explains the experimental settings shared by the three benchmarks, rather than the generic example-file defaults. The frozen Qwen3.6–35B-A3B model serves as both executor and editor. Thus, patch generation does not call a stronger external model.

Proposal budgets and evidence construction. For mixed-outcome groups, successful and failed cases are defined by the benchmark's binary outcome, not by advantage sign alone. Ordinary queries are ranked globally by absolute advantage. A successful positive-advantage anchor is paired with up to two failed same-task peers; a failed negative-advantage anchor is paired with up to two successful peers. Peers are also prioritized by $|A|$. All-failed groups are processed separately: unequal-reward failures can support a shared-failure diagnosis, while up to two reserved queries cover equal-reward failures with zero advantage. These exploration queries are selected across tasks within the same six-query budget. They may propose a corrective rule but cannot by themselves create a signed replay preference. All-success groups supply no corrective query. With six requests and at most two patches per request, a proposal call can produce up to twelve patches, not merely two in total. Duplicate or invalid edits are filtered before replay.

Replay safeguards and stopping. The screening subset contains up to eight scoreable trajectories, ranked by $|A|$ and then by reward deficit when magnitudes tie. The current skill is included in screening, and the top two entries are shortlisted for full replay; the current skill is always scored on the full batch even if not shortlisted. Every candidate is compared with the skill that collected the batch through the same teacher-forcing path. This canonical reference avoids numerical differences between rollout-time decoding scores and replay-time scores being mistaken for a skill change. The reference remains fixed through all local updates. Replay uses raw model likelihoods; when non-neutral sampling filters or penalties are enabled, these ratios are not exact ratios of the transformed decoding distribution.

Table 5: SKILLGRACE hyperparameters and their roles. Candidate and query limits apply to each proposal call within a replay update.

Hyperparameter	Value	Meaning
Experience collection and replay		
Outer iterations	30	Number of online task batches.
Tasks per batch	4	Maximum distinct tasks in a batch.
Rollouts per task	8	Repeated executions forming one task group.
Rollout workers	8	Concurrent execution workers; not extra samples.
Replay updates per batch	3	Maximum accepted-update attempts on one batch.
Patches per evidence query	2	Maximum independent edits requested per query.
Replay screening subset	8	Highest-$\lvert A\rvert$ scoreable trajectories.
Full-buffer finalists	2	Highest-ranked screening entries rescored in full.
Ratio clip ε	0.2	Surrogate ratio bounds $[0.8, 1.2]$.
Maximum empirical KL	0.03	Cumulative shift budget from the behavior skill.
Minimum relative ESS	0.4	Reject concentrated likelihood-ratio weights.
Maximum clipped fraction	0.3	Reject excessive out-of-range token ratios.
Minimum surrogate gain	0	Required gain over the unchanged current skill.
Complexity weight	0	No edit-size penalty in the replay objective.
Deficit trigger	1.0	Skip editing below this total reward deficit.
Patch content limit	1,000	Maximum inserted word-like units, not model tokens.
Objective token scope	Visible action	Score public assistant content only.
Advantage stabilizer	10^{-8}	Constant added to the within-group deviation.
Advantage magnitude cap	5	Clip normalized advantages to $[-5, 5]$.
Patch editor		
Maximum generated tokens	50,000	Completion budget per editor request.
Temperature	1.0	Sampling logit temperature.
Top-p / top-k / min-p	$0.95/20/0$	Nucleus and top-20 filtering; min-p disabled.
Presence / repetition penalty	$1.5/1.0$	Penalize previously generated tokens; neutral repetition multiplier.
Evidence character budget	24,000	Maximum serialized evidence size per query.
Reasoning excerpt limit	6,000	Maximum characters per permitted reasoning excerpt.
Evidence records per query	3	Anchor plus at most two peers in a contrast.
Evidence queries per proposal	6	Maximum independent editor evidence requests.
Zero-advantage exploration	2	Reserved queries for equal-reward all-failed groups.
Random seed	42	Reproducible task order, rollout, and editor seeds.

In addition to the clipped surrogate, acceptance checks the empirical KL proxy, relative effective sample size (ESS), and clipped fraction. If Z is the number of selected visible tokens and ρ_j their likelihood ratios, the implementation computes

$$\widehat{D} = \max\left(0, -\frac{1}{Z}\sum_{j=1}^{Z}\log\rho_j\right), \quad \mathrm{rESS} = \frac{(\sum_j \rho_j)^2}{Z\sum_j \rho_j^2}, \quad f_{\mathrm{clip}} = \frac{1}{Z}\sum_j \mathbf{1}[\rho_j \notin [1-\varepsilon, 1+\varepsilon]].$$

Candidates must satisfy $\widehat{D} \le 0.03$, $\mathrm{rESS} \ge 0.4$, and $f_{\mathrm{clip}} \le 0.3$. The sampled KL proxy is a local diagnostic, not a guarantee of performance preservation on unseen tasks. ESS and clipped fraction guard against reusing experience after the candidate has moved too far from the collecting policy. The selected candidate must also clear the gain margin against the current skill. Replay stops when the current skill wins, no valid candidate is available, no trajectory is scoreable, or the update limit is reached. Before editing, the total deficit $\sum_{i,k}(1 - R^{\mathrm{train}}_{i,k})$ is checked; a value below 1.0 skips proposal and replay for that batch.

Training reward and reporting convention. For rollout k of task i, each benchmark adapter returns a bounded training reward $R^{\mathrm{train}}_{i,k}$ and a binary hard reward $R^{\mathrm{hard}}_{i,k}$. We use a soft training reward for SearchQA and SpreadsheetBench to distinguish partially correct rollouts, and a binary completion reward for evaluable ALFWorld episodes. The group-relative advantages used for patch

generation and replay are computed from $R^{\text{train}}_{i,k}$, not from $R^{\text{hard}}_{i,k}$. Because held-out evaluation uses one rollout per task, the reported benchmark score is $100N^{-1}\sum_{i=1}^{N} R^{\text{hard}}_{i,1}$ over the N test tasks. Across benchmarks, a terminal generation failure receives $R^{\text{train}}_{i,k} = -1$ and $R^{\text{hard}}_{i,k} = 0$. The generation-failure flag includes a completion ending with `finish_reason=length`, an empty or reasoning-only response, or a failure to align the public action tokens reliably. It is distinct from an otherwise evaluable but incorrect answer or incomplete episode, whose ordinary training reward is determined by the benchmark. The benchmark-specific definitions follow.

Adapter completion budgets. The benchmark configuration `max_completion_tokens` overrides the generic executor completion budget for each adapter call. We use 4,096 for SearchQA, 4,096 for each ALFWorld action call, and 8,000 for each SpreadsheetBench generation or repair call. These limits apply per model completion, including generated reasoning, rather than to the visible tokens scored by replay or to the entire multi-turn episode. The editor has its own 50,000-token completion budget. ALFWorld additionally has a 50-step episode limit, and SpreadsheetBench allows at most three generation rounds. An episode that exhausts its environment-step budget is an ordinary failure unless a terminal generation failure is also recorded; the two budgets should not be conflated.

SearchQA adapter. Each rollout is a single model call. The user message contains the question and up to 6,000 characters of retrieved context, and requests a short answer inside an explicit answer tag. Generation is capped at 4,096 tokens. Let $\widehat{y}_{i,k}$ be the extracted answer and $\mathcal{Y}_i$ the set of acceptable answers. We lowercase answers, remove punctuation and articles, and normalize whitespace. The training reward is the maximum token-overlap F1 across references, $R^{\text{train}}_{i,k} = \max_{y\in\mathcal{Y}_i} \mathrm{F1}_{\text{tok}}(\widehat{y}_{i,k}, y)$, whereas $R^{\text{hard}}_{i,k} = 1$ only when the normalized prediction exactly matches a normalized reference. We therefore report normalized exact-match accuracy as a percentage. No search tool or external environment is invoked.

SpreadsheetBench adapter. The adapter presents the natural-language instruction, the required answer range, a bounded preview of the input workbook, and a contract requiring a reusable Python program. Each task runs in a fresh temporary workspace. The same generated program is executed against every workbook associated with the task, and the agent may repair missing or failing code using bounded execution feedback for at most three rounds. Each generation is capped at 8,000 tokens, and each program execution has a 600-second timeout. The evaluator compares only the designated cells with the reference workbook. If task i contains C_i workbook cases and $q_{i,k,c}$ indicates whether case c passes in rollout k, then the training reward is the passed-case fraction $R^{\text{train}}_{i,k} = C_i^{-1}\sum_{c=1}^{C_i} q_{i,k,c}$. A rollout is successful only when every case passes, so $R^{\text{hard}}_{i,k} = \mathbf{1}[\sum_c q_{i,k,c} = C_i]$. We report the percentage of test tasks satisfying this all-cases criterion.

ALFWorld adapter. Each rollout begins from a fresh text-environment instance. At every step, the adapter supplies the goal and accumulated observations, and the model emits one action in a fixed action tag. That action is executed in ALFWorld, and the next observation is appended to the conversation. This interaction continues until the goal is completed, the episode terminates, or the 50-step budget is reached; each action call is capped at 4,096 generated tokens. Binary task completion is read from the environment's `won` flag. For an evaluable episode, it is used as both the training reward and hard outcome: $R^{\text{train}}_{i,k} = R^{\text{hard}}_{i,k} = \mathbf{1}[\texttt{won}_{i,k}]$. A terminal generation failure is the exception, receiving $R^{\text{train}}_{i,k} = -1$ while remaining unsuccessful with $R^{\text{hard}}_{i,k} = 0$. The environment's partial goal-condition score is retained for diagnostics but does not affect training. We report the percentage of episodes completed within the step budget. Thus, only rollout collection advances the environment, whereas patch generation and replay operate entirely on the stored interaction history.

C Prompt Templates

This section reports the prompts used by the benchmark adapters and patch editor. Each prompt is shown in a separate box. Text enclosed by angle brackets denotes content inserted at run time, such as the current skill, task instance, execution feedback, or serialized trajectory evidence. For readability, the boxes replace these variable fields with placeholders while preserving the operative

instructions and output contracts. Replay selection does not use a generation prompt: it teacher-forces the stored trajectories under each candidate skill. The colored title strips distinguish executor requests, editor instructions, and evidence schemas; explanatory paragraphs outside the boxes are not part of the prompts sent to the model.

Shared skill injection. For every benchmark, the current skill is appended exactly once to the benchmark-specific system message.

Shared executor system-message template

```
<BENCHMARK SYSTEM INSTRUCTION>

## Skill
<CURRENT SKILL>
```

SearchQA. The retrieved context is the same bounded input seen by the executor and later shared with the editor. Reference answers are used only by the evaluator, not inserted into this request. The answer tag defines the adapter's extraction contract; visible response tokens, rather than parser-separated reasoning, are eligible for replay scoring.

SearchQA executor request

```
System: You are a grounded question-answering agent. Use only the
supplied context, reason carefully, and keep the final answer concise.

## Skill
<CURRENT SKILL>

User:
## Context
<RETRIEVED CONTEXT>

## Question
<QUESTION>

## Output Format
Return the final answer inside <answer>...</answer>.
```

SpreadsheetBench. The input and output paths are injected by the harness for each workbook case. The preview supplies context, not a complete replacement for reading the workbook. Repair turns reuse the conversation and provide bounded execution errors, not the reference workbook or correct answer cells.

SpreadsheetBench executor request

```
System: You solve spreadsheet manipulation tasks by writing a general
Python program. Follow the execution contract exactly and preserve
workbook content that the instruction does not ask you to change.

## Skill
<CURRENT SKILL>

User:
## Instruction
<SPREADSHEET INSTRUCTION>

## Instruction Type
<INSTRUCTION TYPE>

## Answer Position
<ANSWER RANGE>

# Input spreadsheet preview
<BOUNDED WORKBOOK PREVIEW>

## Execution Contract
Write one reusable Python program using openpyxl. The harness injects
INPUT_PATH and OUTPUT_PATH. Load INPUT_PATH, perform the requested
```

```
manipulation, and save the result to OUTPUT_PATH. Do not hardcode
values from this preview because the same program is evaluated on
every workbook case.  The preview may be truncated; do not hardcode
row counts or assume the data ends at the last previewed row.  Iterate
over all actual rows in the workbook instead.  Return only one complete
Python code block.
```

SpreadsheetBench repair request

```
The program could not be executed successfully.  Repair the program
and return a complete replacement Python code block.  Do not hardcode
workbook values.

Execution feedback:
<EXECUTION ERROR>
```

ALFWorld. Each rollout owns a fresh environment instance. The continuation template is appended after the preceding action and observation; it does not restart the episode. The adapter extracts the public action and passes it to the environment, while retaining the full interaction for later replay.

ALFWorld initial executor request

```
System:  You are controlling an ALFWorld text environment.  At each
turn, choose exactly one valid environment action.

## Skill
<CURRENT SKILL>

User:
## Task Goal
<TASK GOAL>

## Initial Observation
<INITIAL OBSERVATION>

## Output Format
Return one action inside <action>...</action>.
```

ALFWorld continuation request

```
## Environment Observation
<LATEST OBSERVATION>

Return the next action inside <action>...</action>.
```

Same-task contrastive patch editor. The production editor receives one anchor trajectory and its same-task peers. The common request below is instantiated with the appropriate objective shown in the following boxes. The query type and explicit outcome labels establish the direction of comparison. Absolute advantages organize and prioritize the evidence upstream; the editor does not infer success from advantage sign. Each query is submitted independently with the exact current skill and its version. Patches must cite their supporting evidence and are not ranked by the editor.

Same-task patch-editor request

```
System:  You revise reusable Markdown instructions for a task-solving
language model using observed rollouts.  Make only evidence-based
changes, follow the requested patch schema exactly, and return the
final answer only as one JSON object.

User:  You revise a reusable Markdown skill that a task-solving
language model reads before it acts.  The model's weights do not
change; improvement comes only from editing this skill.  Do not solve
the task or grade the rollouts.  Propose small instruction changes that
could improve behavior on future tasks.
```

```
Query type:  <QUERY TYPE>
Task ID: <TASK ID>
Focal evidence ID: <ANCHOR ID>

Evidence outcomes assigned by the benchmark evaluator:
<SUCCESS AND FAILURE LABELS>

Objective for this evidence query:
<QUERY-SPECIFIC OBJECTIVE>

Use this procedure:
1.  Read the current skill before judging the rollouts.
2.  Analyze only this independent evidence case.  Do not infer
information from another editor request.
3.  Compare supplied reasoning excerpts, public outputs or actions, and
observable results according to the outcome labels.
4.  Identify a concrete behavior that helped, hurt, or was consistently
missing.
5.  Check whether the skill already states that behavior clearly.  Do
not propose a duplicate or cosmetic rewrite.
6.  Choose the smallest operation that expresses each justified
improvement as a reusable instruction.
7.  Verify the exact target, patch ID, parent version, and evidence
citations.

How to read the evidence:
All records belong to one task.  For contrastive queries, the first
record is the anchor and the others are same-task comparators.  The
task field contains non-label task input; its truncated flag identifies
an excerpt.  Reasoning excerpts are diagnostic model behavior, not
ground truth.  Verify their claims against observable results.
Evaluation fields, if supplied by the adapter, contain post-rollout
benchmark feedback.  Do not reconstruct missing reasoning.

Generate at most <MAX CANDIDATES> independent atomic patches.  Use
the shared patch-output contract below.  For a contrastive query,
each patch must cite the anchor and at least one comparator.  For an
all-failed query, cite the supplied records that support the diagnosis.
Do not rank candidates or combine unrelated lessons.  Never copy a task
answer, proper name, or task-specific constant into the skill.  If the
evidence is inconclusive or the skill already provides the instruction,
return {''patches'':[]}.  Return JSON only.

Current skill:
<CURRENT SKILL>

Same-task evidence query:
<SERIALIZED TASK, TRAJECTORIES, AND EVALUATION EVIDENCE>
```

Positive-contrast objective

```
The focal evidence is the successful case and every comparator is a
failed case.  Explain why the successful reasoning and final behavior
outperformed the failures.  Identify the decisive behavior that
the successful case performed correctly, then preserve, clarify, or
generalize it only when it is missing or unclear in the current skill.
```

Negative-contrast objective

```
The focal evidence is the failed case and every comparator is a
successful case.  Explain why the focal reasoning and final behavior
had lower performance than the successes.  Identify what it missed,
misread, reversed, or executed incorrectly, then propose one general
repair when the current skill does not already provide it.
```

All-failed objective

```
Every supplied rollout attempted the same task and failed.  There is no
successful comparator.  Diagnose the likely reason for each failure
from its supplied reasoning and final behavior, then identify any
shared failure pattern, missing prerequisite, repeated ineffective
behavior, or absent policy instruction.  Return no patch when the
evidence does not justify a general change.
```

Zero-advantage all-failed exploration objective

```
Every supplied trajectory attempted the same task and failed with the
same group-relative return.  There is no successful comparator and no
relative advantage signal.  Detect the likely reason each trajectory
failed.  Compare the observable trajectories for a shared failure
pattern, missing prerequisite, repeated ineffective behavior, or
absent skill instruction.  Do not claim that the exploration anchor
outperformed another trajectory.  Return an empty patch list when no
general change is justified.
```

Evidence assembly and serialization. The templates below expose how an editor request is constructed, rather than only showing a placeholder for its evidence. A query contains at most three complete observable trajectories. System messages and duplicated initial task messages are removed from the editor projection because the current skill and shared task are supplied separately. Repeated copies of an immediately preceding observation are also removed. Observable actions and results retain their order. SearchQA and SpreadsheetBench may include bounded, parser-separated reasoning excerpts; ALFWorld supplies its observable action–observation trace. The 24,000-character budget applies to serialized evidence, not to the entire editor prompt. If necessary, task context becomes an explicitly marked head-and-tail excerpt and reasoning excerpts shrink uniformly. Comparator actions and observable results are not silently dropped to fit the budget. A query is skipped if its mandatory evidence still cannot fit.

Same-task evidence construction template (preprocessing)

```
Input:  stored experience, current skill, query budget, record budget.

Group records by task ID and read their evaluator-assigned binary
outcomes.
Rank eligible ordinary anchors globally by absolute advantage.
For a successful positive-advantage anchor, retrieve failed same-task
peers.  For a failed negative-advantage anchor, retrieve successful
same-task peers.  Rank peers by absolute advantage and keep at most
two.
Handle all-failed groups separately and reserve at most two requests
for equal-reward failures without claiming a relative ranking.
Keep at most six queries in total.  Serialize each independently with
its outcome labels, query objective, exact current skill, and allowed
patch IDs.
```

Serialized same-task evidence template

```
{``task'':  {``content'':  ``<SHARED TASK INPUT>'', ``truncated'':
<BOOLEAN>},
``evidence'':  [
  {``evidence_id'':  ``<ANCHOR ID>'',
    ``evaluation'':  <OPTIONAL ADAPTER FEEDBACK>,
    ``trajectory'':  [<OPTIONAL REASONING EXCERPT>,
      {``role'':  ``assistant'', ``content'':  ``<OUTPUT OR ACTION>''},
      {``role'':  ``environment'', ``content'':  ``<OBSERVATION>''},
      <REMAINING OBSERVABLE EVENTS>]},
  {``evidence_id'':  ``<COMPARATOR ID>'',
    ``trajectory'':  [<SAME-TASK OBSERVABLE EVENTS>]},
  <OPTIONAL SECOND COMPARATOR>
```

```
]}
```

The `evaluation` field is omitted when the adapter supplies no approved feedback. Event roles may also be `tool`, `evaluator`, or a genuine later `user` turn. The outcome labels appear separately in the editor request; numeric advantages select the evidence but need not be serialized into these records. The immutable full trajectories are retained for replay even when the editor projection is shorter.

Shared patch-output contract

```
Return {``patches'': [<PATCH OBJECTS>]} or {``patches'': []}.

Each object contains patch_id, parent_version, operation, and
evidence_ids. Use only the supplied IDs, in order, and the exact
current parent version.
append: add a missing instruction through content.
insert_after: copy an exact unique target and insert content after it.
remove: copy the exact unique target to remove.
replace: copy the exact old target and provide replacement content and
new_rule in the form ``When <condition>, <behavior>.''

Each patch changes one coherent Markdown block and one general
behavior. Do not copy answers, proper names, or task-specific
constants. Contrastive patches cite the anchor and a comparator;
cross-task patches cite every supplied evidence ID. An empty patch
list is valid.
```

Cross-task ablation editor. The cross-task condition changes only the editor evidence. It submits separate success-only and failure-only requests whose records come from different tasks; rollout collection, replay selection, trust-region checks, and evaluation are unchanged. Each request contains two or three distinct tasks with the same outcome; successful and failed records never share a prompt. Records are ranked by $|A|$ within tasks, each record is used at most once per proposal call, and success and failure queries alternate within the six-query budget. A singleton is not submitted. Because there is no shared task input, each serialized record carries its own task ID and context. The cross-task arm does not use reserved zero-advantage exploration slots.

Cross-task success editor request

```
System: Revise reusable Markdown instructions using only the supplied
evidence. Return one JSON object.

User: You revise a reusable Markdown skill read by a task-solving
language model. Model weights do not change. Analyze only the
supplied observable rollouts and propose small, general instructions
for future tasks.

Study condition: cross_task_outcome
These are successful rollouts from DIFFERENT tasks. Identify a
general successful strategy shared across them. Do not compare either
rollout with a failure: none is supplied. All supplied trajectories
succeeded.

Read the current skill before proposing a change. Treat assistant
reasoning as diagnostic behavior, not ground truth. Each task context
belongs only to its own evidence record. Never copy an answer,
task-specific fact, or proper name into the skill. Return at most
<MAX CANDIDATES> independent atomic patches and cite every supplied
evidence ID. If no shared lesson is justified, return {``patches'':[]}.
Return JSON only.

Current skill:
<CURRENT SKILL>

Evidence:
<SUCCESSFUL TRAJECTORIES FROM DIFFERENT TASKS>
```

Cross-task failure editor request

```
System: Revise reusable Markdown instructions using only the supplied
evidence. Return one JSON object.

User: You revise a reusable Markdown skill read by a task-solving
language model. Model weights do not change. Analyze only the
supplied observable rollouts and propose small, general instructions
for future tasks.

Study condition: cross_task_outcome
These are failed rollouts from DIFFERENT tasks. Identify a shared,
generalizable failure pattern and a repair. Do not infer what a
successful rollout would have done. All supplied trajectories failed.

Read the current skill before proposing a change. Treat assistant
reasoning as diagnostic behavior, not ground truth. Each task context
belongs only to its own evidence record. Never copy an answer,
task-specific fact, or proper name into the skill. Return at most
<MAX CANDIDATES> independent atomic patches and cite every supplied
evidence ID. If no shared lesson is justified, return {''patches'':[]}.
Return JSON only.

Current skill:
<CURRENT SKILL>

Evidence:
<FAILED TRAJECTORIES FROM DIFFERENT TASKS>
```

Serialized cross-task collections template

```
{''evidence'': [
  {''evidence_id'': ''<RECORD 1>'', ''task_id'': ''<TASK 1>'',
    ''task'': {''content'': ''<TASK 1 INPUT>'', ''truncated'':
<BOOLEAN>},
    ''trajectory'': [<TASK 1 OBSERVABLE EVENTS>],
    ''evaluation'': <OPTIONAL ADAPTER FEEDBACK>},
  {''evidence_id'': ''<RECORD 2>'', ''task_id'': ''<TASK 2>'',
    ''task'': {''content'': ''<TASK 2 INPUT>'', ''truncated'':
<BOOLEAN>},
    ''trajectory'': [<TASK 2 OBSERVABLE EVENTS>]},
  <OPTIONAL RECORD FROM A THIRD DISTINCT TASK>
]}
```

All records in this template are either successful or failed according to the query type. Their contexts are never presented as if they belonged to one task. The same observable-event projection and evidence-size limits apply to both conditions.