BEYOND INITIAL SKILLS: CONTRAST-GUIDED SKILL EVOLUTION FOR ROBUST LEARNING IN LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model (LLM) agents can learn continually by accumulating and updating external procedural skills while keeping their model parameters frozen. Existing approaches typically evolve these skills by revising prior skills based on interaction experiences collected from new tasks. However, we find that their learning gains vary substantially with initial skills, even under the same sequence of subsequent training tasks, resulting in unstable or even negative skill evolution. Further analysis shows that initial skills influence both the experiences collected through agent interactions and the subsequent skill updates, causing their effects to persist throughout the evolution process. To enable robust skill evolution beyond initial skills, we propose **Skill-CoE**, a **Co**ntrast-guided skill **E**volution framework that explores alternative skill update directions and leverages their complementary strengths for effective updates. Specifically, given the same interaction experience, Skill-CoE constructs two evolution candidates by revising the current skill and reconstructing a new skill, respectively. It then compares their executions on the same tasks using a pairwise preference objective, with the current skill serving as a fixed reference to identify complementary strengths. Guided by these comparisons, Skill-CoE recombines the candidates to incorporate useful new procedures while preserving effective existing experience. Experiments on ALFWorld, AppWorld, and ScienceWorld show that Skill-CoE consistently achieves stable learning gains across all evaluated initial skills and outperforms existing methods in final test performance. Code is available at https://anonymous.4open.science/r/Skill-CoE-2737.

1 INTRODUCTION

Large language model (LLM) agents can solve complex tasks through planning, tool use, and environmental interaction(Yao et al., 2023; Schick et al., 2023; Ichter et al., 2023), while sustained operation further requires learning from past successes and failures to improve future decisions (Yao et al., 2023; Zheng et al., 2025). A promising approach to continual agent learning is to accumulate and update external knowledge from interaction experience while keeping model parameters frozen (Park et al., 2023; Shinn et al., 2023; Yuksekgonul et al., 2025; Xu et al., 2025; Ouyang et al., 2026). Among different forms of external knowledge, skills encode reusable procedural knowledge through applicability conditions, action steps, and execution constraints, allowing agents to retrieve and apply them across tasks (Ding et al., 2026). Early studies focused on accumulating and composing executable skill programs (Wang et al., 2023), while subsequent work explored more abstract and hierarchical skill representations and extracted reusable skills from execution trajectories (Yu et al., 2026; Wang et al., 2026; Ni et al., 2026). More recently, skills themselves have been treated as optimization targets, with execution feedback used to iteratively refine their content and improve subsequent task execution (Ma et al., 2026; Liu et al., 2026; Mi et al., 2026; Yang et al., 2026).

Despite this progress, repeatedly evolving skills from their current state raises an important question about whether subsequent learning depends on where the evolution starts. A related phenomenon has been observed in deep reinforcement learning, where biases induced by early experience can prevent models from effectively incorporating useful evidence encountered later (Nikishin et al., 2022). We therefore ask whether continual learning through external skills is similarly sensitive to skill initialization. To investigate this question, we construct distinct initial skills from training tasks of varying difficulty using a standardized interaction and skill-distillation procedure, and then ex-

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.

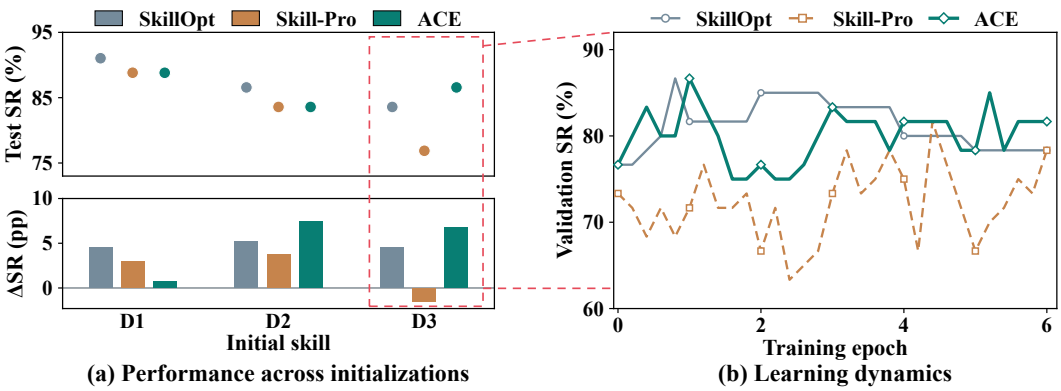


Figure 1: Learning from different initial skills on ALFWorld. **(a)** Final test SR (top) and gains over each method's initial performance (bottom, percentage points). D1–D3 correspond to Easy, Medium, and Hard source tasks (Section 4.1). **(b)** Validation SR during learning from D3.

pose agents initialized with these skills to the same sequence of subsequent training tasks (Detailed in Section 4.3). Figure 1 illustrates the results on the widely used ALFWorld benchmark (Shridhar et al., 2021) for three recent representative methods, SkillOpt (Yang et al., 2026), Skill-Pro (Mi et al., 2026), and ACE (Zhang et al., 2026). As shown in Figure 1(a), existing methods exhibit substantially different test-set learning gains across initial skills, with some even producing negative gains. Figure 1(b) further shows that their validation performance may plateau or fluctuate throughout subsequent learning. These results reveal a clear sensitivity to skill initialization and motivate us to develop a more robust skill evolution approach that can consistently benefit from new experience across different initial skills.

Building on these observations, we further investigate why the influence of initial skills persists throughout subsequent learning. Existing methods typically evolve skills by repeatedly revising the current skill based on newly collected interaction experience. Each revised skill then guides subsequent interactions and serves as the basis for the next update, allowing the influence of initial skills to propagate through successive interaction and update cycles (Ding et al., 2026). Achieving robust skill evolution across different initial skills therefore presents two key challenges. **C1:** How can we explore effective update directions beyond the structure imposed by the current skill? **C2:** As different update directions may improve some behaviors while degrading others, how can we identify their complementary strengths and leverage them for more effective skill updates?

To address these challenges, we propose **Skill-CoE**, a **Co**ntrast-guided skill **E**volution framework for robust skill evolution in LLM agents. To be specific, given the same interaction experience, Skill-CoE first explores two alternative update directions by revising the current skill and reconstructing a new skill directly from experience **(C1)**. It then contrasts their executions on the same tasks to identify complementary behavioral strengths **(C2)**. We adapt a pairwise preference objective from DPO (Rafailov et al., 2023), using the current skill as a fixed reference to quantify the behavioral changes induced by each update direction. The resulting contrasts guide skill recombination to incorporate useful procedures while preserving effective existing behaviors. Finally, extensive experiments on ALFWorld (Shridhar et al., 2021), AppWorld (Trivedi et al., 2024), and ScienceWorld (Wang et al., 2022) demonstrate that Skill-CoE enables robust skill evolution across different initial skills and consistently improves final task performance. Our contributions are summarized as follows:

- We systematically investigate the sensitivity of skill evolution to initial skills, showing that existing methods can yield substantially different, and even negative, learning gains across initial skills despite following the same sequence of subsequent training tasks.
- We propose Skill-CoE, a contrast-guided skill evolution framework that explores alternative update directions beyond the structure of the current skill and contrasts their behavioral changes to identify complementary strengths for effective skill recombination.
- We conduct extensive experiments on ALFWorld, AppWorld, and ScienceWorld, demonstrating that Skill-CoE achieves consistent positive learning gains across different initial skills and outperforms existing methods in final test performance. Further analyses validate the effectiveness of alternative update exploration, preference-based behavioral contrast, and skill recombination.

2 PRELIMINARIES

2.1 AGENT INTERACTION AND SKILL UPDATES

With LLM parameters θ frozen, an agent uses the current skill ω_k to execute a task batch $\mathcal{T}_k$ and updates the skill based on the resulting interaction experience. Each skill encodes procedural knowledge in terms of applicability conditions, action steps, and execution constraints. For each task $x \in \mathcal{T}_k$, the interaction trajectory and the resulting batch experience are defined as:

$$\tau_{k,x} \sim \pi_\theta(\cdot \mid x, \omega_k), \qquad \mathcal{D}_k = (x, \tau_{k,x}, R(\tau_{k,x})) : x \in \mathcal{T}_k, \tag{1}$$

where π_θ denotes the trajectory distribution conditioned on task x and skill ω_k. Each trajectory $\tau_{k,x}$ consists of agent outputs, actions, and environment observations, and $R(\tau_{k,x}) \in \mathbb{R}$ denotes its environment return. The agent then updates the skill as $\omega_{k+1} = \mathcal{U}(\omega_k, \mathcal{D}_k)$, which is used to execute the next task batch. The update operator $\mathcal{U}$ generates, evaluates, and selects candidate skills, retaining ω_k when no candidate update is accepted. Section 3 presents our instantiation of $\mathcal{U}$, while Appendix A provides the complete algorithm and implementation details.

2.2 DIRECT PREFERENCE OPTIMIZATION

Direct Preference Optimization (DPO) (Rafailov et al., 2023) learns a policy from a preference dataset $\mathcal{P}$, where each tuple (x, y^+, y^-) indicates that response y^+ is preferred to y^- for the same input x. Given a trainable policy π_φ and a fixed reference policy π_{ref}, DPO minimizes:

$$\mathcal{L}_{\text{DPO}}(\varphi) = -\mathbb{E}_{(x,y^+,y^-)\sim\mathcal{P}}\left[\log\sigma\left(\beta\log\frac{\pi_\varphi(y^+ \mid x)}{\pi_{\text{ref}}(y^+ \mid x)} - \beta\log\frac{\pi_\varphi(y^- \mid x)}{\pi_{\text{ref}}(y^- \mid x)}\right)\right], \tag{2}$$

where $\sigma(t) = (1+\exp(-t))^{-1}$ is the logistic sigmoid and $\beta > 0$ scales the relative log-probability difference. The objective updates φ to increase the log-probability gap between preferred and dispreferred responses relative to the reference policy. We borrow this objective's pairwise, fixed-reference form to optimize skill text rather than model parameters (Section 3.3.1).

3 METHOD

Figure 2 illustrates the three stages of Skill-CoE. First, contrastive sampling generates revised and reconstructed skills from the same experience to explore alternative update directions. Second, both skills execute the same tasks, and their returns are compared to construct trajectory preferences that capture behavioral differences. Third, contrast-guided recombination uses these preferences to identify complementary strengths and integrate them into a new skill. The executor remains frozen throughout. The complete algorithm and implementation details are provided in Appendix A.

3.1 CONTRASTIVE SAMPLING

Revising the prior skill may inherit its limitations, while reconstructing from experience may lose effective procedures. We therefore explore both update directions using the same experience:

$$\omega^{\text{rev}} = U_{\text{rev}}(\omega_k, \mathcal{D}_k), \qquad \omega^{\text{rec}} = U_{\text{rec}}(\mathcal{D}_k), \tag{3}$$

where U_{rev} revises ω_k using $\mathcal{D}_k$, while U_{rec} reconstructs a skill from $\mathcal{D}_k$ without access to ω_k. Both follow the same task and output format, isolating update direction as their key difference. We then compare their executions to identify the induced behavioral changes.

3.2 CONSTRUCTING WITHIN-TASK TRAJECTORY PREFERENCES

Existing records capture only behaviors under ω_k and provide no outcomes for alternative procedures. We therefore execute both candidates on selected tasks $\mathcal{X}_k \subseteq \mathcal{T}_k$ under the same initial states and interaction budgets. Let $\tau_x(\omega)$ denote the trajectory produced by skill ω, with $\tau_x(\omega_k) = \tau_{k,x}$. We construct within-task preferences based on returns:

$$\mathcal{E}_x = \big\{(x, \tau_x(\omega), \tau_x(\omega')) : \omega, \omega' \in \{\omega_k, \omega^{\text{rev}}, \omega^{\text{rec}}\}, R(\tau_x(\omega)) > R(\tau_x(\omega'))\big\}. \tag{4}$$

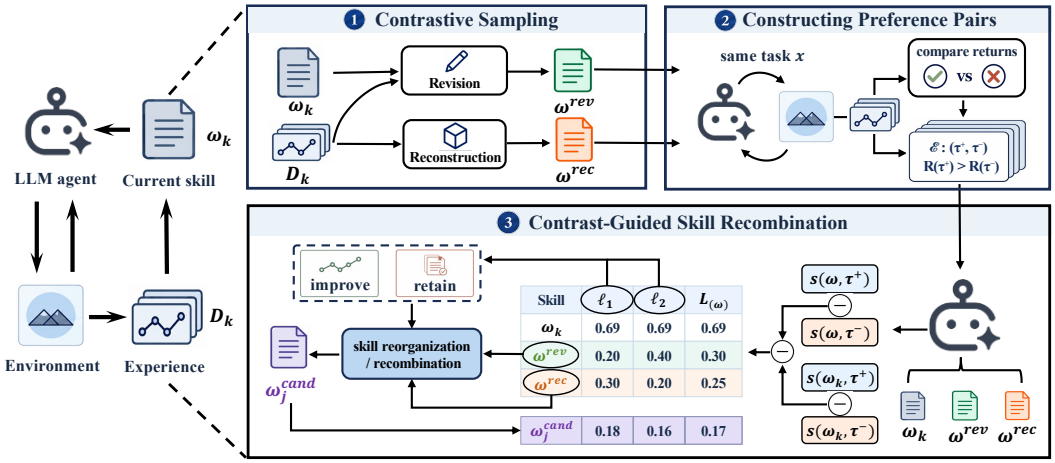


Figure 2: Overview of Skill-CoE. (1) Contrastive Sampling: revise ω_k and reconstruct from $\mathcal{D}_k$, yielding ω^{rev} and ω^{rec}. (2) Constructing Preference Pairs: compare returns on shared tasks to form $\mathcal{E}_k$. (3) Contrast-Guided Skill Recombination: preference losses identify complementary behaviors and guide ω^{cand} generation. Acceptance is detailed in Appendix A.4.

Each pair $e = (x, \tau_e^+, \tau_e^-)$ consists of a preferred and a dispreferred trajectory. We define $\mathcal{X}_k^\Delta = x \in \mathcal{X}_k : |\mathcal{E}_x| > 0$ and aggregate the fixed preference evidence as $\mathcal{E}_k = \bigcup_{x \in \mathcal{X}_k^\Delta} \mathcal{E}_x$. Comparisons against the current skill reveal potential improvements or behaviors to preserve, while candidate-to-candidate comparisons expose differences between update directions. Ties are excluded, as the preferences compare recorded trajectories rather than rank skills by expected return.

For subsequent recombination, we merge the new and existing records into $\widetilde{\mathcal{D}}_k$ for case retrieval. We also retain successful records from $\mathcal{D}_k$ as replay material $\mathcal{S}_k$ to preserve effective behaviors not captured by return differences (Appendix A.1).

3.3 Contrast-Guided Skill Recombination

No single candidate may capture all effective behaviors revealed by the preference evidence. We therefore evaluate each candidate on the fixed preferences to select a recombination base, and use pairwise differences to identify complementary strengths and their supporting cases. These cases guide targeted recombination, which is further evaluated on the same preference evidence. We use j to index recombination attempts and omit k from the local notation.

3.3.1 Skill Evaluation from Trajectory Preferences

Recorded-action scores. To evaluate candidates without re-executing each one, we replace only the skill text and score recorded actions under fixed histories. For trajectory $\tau = (z_n)_{n=1}^{T_\tau}$ on task x, let $\mathcal{I}_\tau$ contain action or code token positions. We define

$$s(\omega; \tau) = \frac{1}{|\mathcal{I}_\tau|} \sum_{n \in \mathcal{I}_\tau} \log P_\theta(z_n \mid x, \omega, z_{<n}). \tag{5}$$

All candidates use identical scoring positions and recorded prefixes, including preceding actions, observations, and explanatory text. Only action or code tokens contribute to the score, focusing the comparison on execution choices.

Preference objective. A single trajectory score does not express preference for higher-return behavior. We compare preferred and dispreferred scores and subtract the corresponding gap under the fixed skill ω_k, so all candidates are evaluated relative to the same pre-update state. Borrowing the logistic form of DPO with scale fixed to 1, we define

$$\ell_e(\omega) = -\log \sigma\big([s(\omega; \tau_e^+) - s(\omega; \tau_e^-)] - [s(\omega_k; \tau_e^+) - s(\omega_k; \tau_e^-)]\big). \tag{6}$$

A positive argument to σ indicates an increased score advantage for the preferred trajectory. The loss gives diminishing benefit to large improvements while continuing to penalize regressions. We then average within each task and across tasks so that tasks with more preference pairs do not receive greater total weight:

$$\mathcal{L}(\omega) = \frac{1}{|\mathcal{X}_k^{\Delta}|} \sum_{x \in \mathcal{X}_k^{\Delta}} \frac{1}{|\mathcal{E}_x|} \sum_{e \in \mathcal{E}_x} \ell_e(\omega). \tag{7}$$

The aggregate loss selects the recombination base, while pairwise losses identify local strengths. Both measure relative action support under fixed histories and serve as search surrogates rather than standard DPO parameter optimization.

3.3.2 Selecting Skills and Cases for Recombination

Recombination base. From the initial pool $\mathcal{C}_1 = \{\omega_k, \omega^{\mathrm{rev}}, \omega^{\mathrm{rec}}\}$, we select $\omega^{\mathrm{base}} \in \arg\min_{\omega \in \mathcal{C}_1} \mathcal{L}(\omega)$. Keeping this base fixed gives all editing attempts the same starting text and improvement criterion. Its role differs from that of ω_k, the scoring reference in Equation (6).

Complementary donor. A skill with higher overall loss may still outperform the base on individual pairs. For $e \in \mathcal{E}_x$, define its weighted local advantage as $d_e(\omega) = [\ell_e(\omega^{\mathrm{base}}) - \ell_e(\omega)]/(|\mathcal{X}_k^{\Delta}|\,|\mathcal{E}_x|)$, using the task weights in Equation (7). Positive d_e favors the candidate; negative d_e favors the base. To identify useful local strengths without canceling them against weaknesses elsewhere, we sum only positive advantages:

$$G(\omega) = \sum_{e \in \mathcal{E}_k} [d_e(\omega)]_+, \qquad [a]_+ = \max(a, 0). \tag{8}$$

Let $\mathcal{C}_j$ contain the initial skills and complete candidates retained from earlier unsuccessful attempts. We choose $\omega_j^{\mathrm{donor}}$ as the skill with the largest $G(\omega)$ in this pool, excluding the base.

Behavioral cases. We sample improvement and preservation cases in proportion to the donor's positive and negative d_e magnitudes, respectively, and add successful replay to cover behaviors omitted by preference pairs. Sampling details are provided in Appendix A.2.

3.3.3 Candidate Generation and Search

Behavioral comparisons do not directly specify text edits. Consistent with feedback-driven text optimization (Pryzant et al., 2023), we therefore analyze improvement, preservation, and replay cases to obtain findings B_j, then align them with the base and donor rules to obtain M_j. We form the plan $\mathcal{P}_j = U_{\mathrm{plan}}(B_j, M_j, f_{j-1})$ using feedback from the previous attempt. To limit unrelated changes, the plan guides local edits to the fixed base:

$$\omega_j^{\mathrm{cand}} = \mathrm{Apply}\big(\omega^{\mathrm{base}}, U_{\mathrm{edit}}(\omega^{\mathrm{base}}, \mathcal{P}_j)\big)\,. \tag{9}$$

Starting with $f_0 = \varnothing$, we score candidates on $\mathcal{E}_k$ and accept the first satisfying $\mathcal{L}(\omega_j^{\mathrm{cand}}) < \mathcal{L}(\omega^{\mathrm{base}})$. Nonimproving candidates enter the donor pool because they may retain local strengths. Their edits and loss changes inform the next attempt. Each attempt edits the same base rather than accumulating prior edits. After at most J attempts, the search returns an improved candidate or, if none is found, $\omega^{\mathrm{out}} = \omega^{\mathrm{base}}$. Appendix A.3 details generation stages and failure handling.

4 Experiments

4.1 Experimental Setup

Benchmarks and metrics. We evaluate our method on three established benchmarks spanning diverse interactive environments and task structures. **ALFWorld** (Shridhar et al., 2021) involves household object manipulation through text-based actions. We report task success rate (SR). **AppWorld** (Trivedi et al., 2024) requires code execution and API calls to complete tasks across applications. We report task goal completion (TGC) and scenario goal completion (SGC), which requires success on all task variants within a scenario. **ScienceWorld** (Wang et al., 2022) covers 30 types of scientific tasks in a text-based environment involving object manipulation, observation, and experimentation. We report the average environment score (AS) and success rate (SR).

Table 1: Test performance after learning from training tasks without pre-existing skills. Steps denote the average number of interaction steps. Bold marks the best result in each column.

	ALFWorld		AppWorld			ScienceWorld		
Method	SR ↑	Steps ↓	TGC ↑	SGC ↑	Steps ↓	AS ↑	SR ↑	Steps ↓
Non-learning baseline								
Static	67.16	26.75	67.86	46.43	24.15	56.09	38.38	18.17
Trajectory-based memory methods								
Memory	76.12	24.34	76.19	57.14	21.86	60.25	42.80	17.45
RAG	79.10	23.47	77.38	60.71	20.74	61.42	44.28	16.93
External knowledge update methods								
ExpeL	82.84	22.11	80.36	64.29	18.18	67.46	46.49	16.11
ACE	87.31	**13.08**	85.12	73.21	14.89	76.14	59.04	14.99
Skill optimization methods								
SkillOpt	88.81	13.72	83.93	71.43	17.06	70.11	52.77	13.45
Skill-Pro	88.81	18.01	81.55	67.86	20.58	65.86	48.34	18.64
Skill-CoE	**94.78**	14.76	**88.10**	**76.79**	**12.08**	**80.78**	**63.47**	**12.29**

Backbone models. Our main experiments use the frozen Qwen3.8-27B for task execution and external knowledge updates. Agents alternate planning, actions, and environment feedback. We additionally evaluate Skill-CoE with Qwen3.5-9B and Ministral-3-8B-Instruct to assess its effectiveness across backbone models.

Baselines. We compare Skill-CoE with three baseline groups: **(1) Trajectory-based memory methods.** Memory places the session history in context for in-context learning, truncating the oldest content when the context limit is exceeded. RAG retrieves snippets from past trajectories. **(2) External knowledge update methods.** ExpeL (Zhao et al., 2024) extracts rules from experience, while ACE (Zhang et al., 2026) updates knowledge through reflection and curation. We adapt both to batch updates while retaining their knowledge update mechanisms. **(3) Skill optimization methods.** SkillOpt (Yang et al., 2026) optimizes skill documents, while Skill-Pro (Mi et al., 2026) evolves a structured skill pool. A non-learning agent with a fixed configuration (Static) serves as a reference.

Initialization and learning protocol. ALFWorld and AppWorld use either no pre-existing skills or skill libraries constructed from source tasks; ScienceWorld uses no pre-existing skills. Our main comparison uses initial skills from Easy, Medium, and Hard source-task groups, denoted D1–D3, with the same extraction procedure and knowledge budget. For robustness experiments, we also evaluate random equal-sized partitions (R1–R3) and task-scenario or task-type groups (T1–T3) in selected settings. Source tasks are disjoint from training, validation, and test tasks. Each learning run comprises five training batches. Within each environment, initialization conditions share the same subsequent training tasks and batch partitions. External knowledge is frozen during testing. Appendix B details library construction, robustness settings, and task splits.

4.2 Learning from Empty Skill Libraries

Skill-CoE outperforms existing learning methods. Starting without pre-existing task-specific skills, Skill-CoE achieves the best task performance across all three benchmarks (Table 1). It exceeds the strongest baseline by 5.97 percentage points in ALFWorld SR, 2.98 points in AppWorld TGC, and 4.64 points in ScienceWorld AS, while also leading in AppWorld SGC and ScienceWorld SR. It also uses the fewest interaction steps on AppWorld and ScienceWorld.

How experience is updated matters. These results suggest that effective learning depends on how experience is represented and revised. Trajectory-based memory can retain redundancy and noise, making useful information harder to reuse. External knowledge methods may lose useful content through repeated rewriting without sufficient structural constraints and edit boundaries. Skill optimization provides more structure, but improvement can remain constrained by skills formed early in learning. Skill-CoE explores alternative updates while preserving effective behaviors. It

Table 2: Final performance (%) and learning gains (Δ, percentage points) across initial skills. Mean Δ averages gains over D1–D3. Bold marks the best results, including ties.

Method	ALFWorld							AppWorld						
	D1		D2		D3		Mean Δ	D1		D2		D3		Mean Δ
	SR	Δ	SR	Δ	SR	Δ		TGC	Δ	TGC	Δ	TGC	Δ	
ExpeL	88.81	+2.99	79.10	−1.49	82.84	+3.73	+1.74	82.74	**+7.74**	80.36	+2.38	80.95	0.00	+3.37
ACE	88.81	+0.75	83.58	**+7.46**	86.57	+5.23	+4.48	82.14	+2.38	78.57	−0.60	84.52	+4.17	+1.98
SkillOpt	91.04	+4.48	86.57	+5.22	83.58	+4.48	+4.73	83.33	+5.36	77.38	−4.17	85.71	+4.76	+1.98
Skill-Pro	88.81	+2.99	83.58	+3.73	76.87	−1.49	+1.74	82.14	+1.78	81.55	+1.19	80.36	+3.57	+2.18
Skill-CoE	**93.28**	**+6.72**	**90.30**	**+7.46**	**91.79**	**+9.70**	**+7.96**	**83.93**	+3.57	**82.74**	**+2.98**	**87.50**	**+6.55**	**+4.37**

Table 3: Ablation study. Blue subscripts show SR/TGC drops from the full method in points.

Method	ALFWorld SR	AppWorld TGC	AppWorld SGC
Skill-CoE	**94.78**	**88.10**	**76.79**
DR	87.31$_{-7.46}$	83.93$_{-4.17}$	69.64$_{-7.15}$
UP	90.30$_{-4.48}$	82.74$_{-5.36}$	67.86$_{-8.93}$
NR	86.57$_{-8.21}$	80.36$_{-7.74}$	66.07$_{-10.72}$
RR	89.55$_{-5.22}$	82.74$_{-5.36}$	69.64$_{-7.15}$

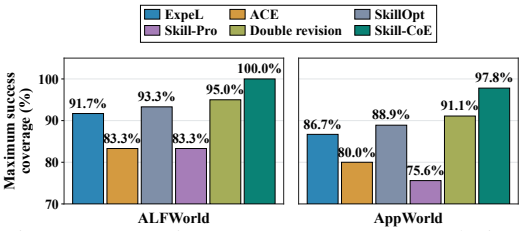


Figure 3: Maximum success coverage during learning. A task is covered if any skill version succeeds at least once.

combines revision with independent reconstruction and uses execution comparisons to guide what to incorporate and retain. The following analyses examine the contributions of these designs.

4.3 Sensitivity to Skill Initialization

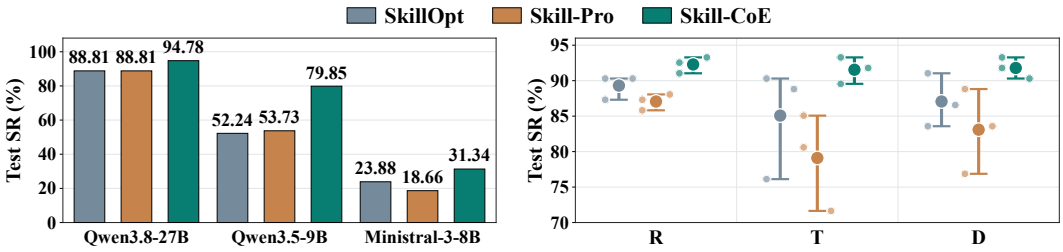
Skill-CoE learns robustly across initial skills. To assess sensitivity to initial skills, we compare learning gains across three initial skill configurations while keeping subsequent training tasks fixed (Table 2). Existing methods exhibit unstable gains: SkillOpt improves by 5.36 points under D1 on AppWorld but regresses by 4.17 points under D2. In contrast, Skill-CoE achieves positive gains and the highest final performance in all six conditions, with the largest mean gains of 7.96 points on ALFWorld and 4.37 on AppWorld. These results demonstrate that Skill-CoE learns robustly from subsequent experience across different initial skills.

4.4 Ablation Study

Our ablation study reveals complementary contributions of Skill-CoE's core components (Table 3). NR (no recombination) causes the largest performance drop on both benchmarks, highlighting the importance of combining skills beyond selecting a candidate. RR (random recombination) also underperforms Skill-CoE, showing that the benefit depends on how recombination is guided. DR (double revision), which replaces independent reconstruction with a second revision, and UP (unpaired outcomes), which replaces within-task preferences with individual trajectory feedback, both reduce performance. These results support complementary roles for generating useful alternatives and constructing relative evidence to guide incorporation. Appendix B.5 details the variants.

4.5 Skill Evolution Analysis

Contrastive sampling broadens access to successful experience. To examine how contrastive sampling expands learning opportunities, we compare success coverage across methods (Figure 3). Skill-CoE achieves the highest coverage on both benchmarks, reaching 100.0% on ALFWorld and 97.8% on AppWorld, versus 95.0% and 91.1% for double revision. Both variants generate two candidates, suggesting that distinct update directions contribute beyond candidate count. Revision starts from the existing skill and may retain its assumptions and action sequences even when experience supports alternatives. Independent reconstruction instead reorganizes the same experience without



(a) Backbone generalization (b) Initial-skill robustness

Figure 5: Robustness on ALFWorld across (a) backbone models and (b) initial-skill construction schemes. In (b), small dots represent individual configurations, large dots their means, and bars their minimum–maximum ranges. R, T, and D denote random grouping, task-type grouping, and difficulty-based grouping, respectively. Each scheme includes three configurations.

the existing skill, allowing alternative procedures or applicability conditions to emerge. These alternatives can succeed on tasks missed by revision, expanding evidence for subsequent learning.

Within-task preferences promote corrective updates. To understand the role of paired feedback, we examine how skill updates repair failures and introduce regressions (Figure 4). Skill-CoE achieves the highest repair rates on both benchmarks, reaching 77.3% on ALFWorld and 74.2% on AppWorld. Removing within-task preferences substantially reduces repair in both environments. On ALFWorld, regression also increases; on AppWorld, lower regression accompanies much weaker repair. Thus, fewer regressions alone do not imply more effective learning. These results suggest that paired feedback helps turn execution experience into corrective updates. Individual outcomes indicate whether an execution succeeds, but leave the updater to infer which alternative behavior should replace it. Within-task comparisons make this relation explicit under a shared task context, reducing ambiguity from differences across tasks. Comparisons favoring the current skill also provide evidence for preserving existing behaviors. This relative feedback can therefore give updates a clearer direction.

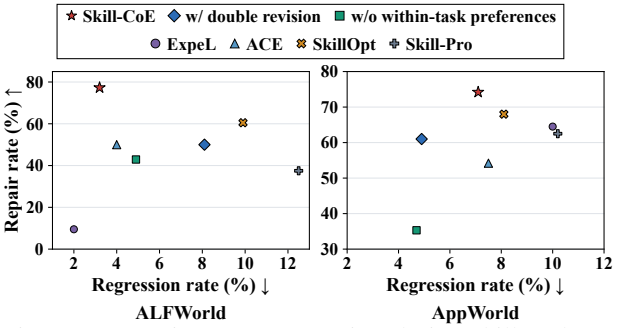


Figure 4: Repair versus regression during skill updates, with higher repair and lower regression being preferable.

Contrast-guided recombination directs skill improvement. We examine whether preference guidance steers successive candidates toward the intended objective (Table 4). Avg. measures the weighted proportion of preference pairs with lower loss than under the preceding candidate; Consist. requires improvement at both consecutive transitions. Among batches reaching three candidates, Skill-CoE leads on both measures across both benchmarks, indicating more persistent progress on the same preference evidence. Pairwise differences connect editing decisions to observed strengths and weaknesses, guiding which behaviors to incorporate or preserve. Random recombination lacks this targeted connection. These results support the role of preference guidance in directing skill updates, while measuring pairwise progress rather than aggregate loss reduction.

Table 4: Preference-pair improvement (%).

	ALFWorld		AppWorld	
Method	Avg. ↑	Consist. ↑	Avg. ↑	Consist. ↑
Skill-CoE	**66.25**	**31.6**	**59.72**	**29.2**
w/ random recombination	43.75	12.5	25.00	0.0

4.6 Robustness Analysis

Robustness across backbone models. Skill-CoE outperforms both baselines across all three backbones, with a particularly large advantage on Qwen3.5-9B (Figure 5a). This suggests that effective skill evolution does not require the strongest backbone, but still depends on sufficient un-

derlying capabilities. A capable model can generate useful alternatives, interpret execution feedback, and translate behavioral comparisons into effective edits. With a weaker backbone, limitations in task execution and skill updating may compound, restricting the benefit of subsequent learning. The lower performance on Ministral-3-8B-Instruct is consistent with this dependence: Skill-CoE remains beneficial, but external skill updates cannot fully compensate for limited model capabilities.

Robustness to initial-skill construction. To test whether Skill-CoE depends on an initialization scheme, we construct initial skills through random, task-type, and difficulty-based grouping (Figure 5b). Skill-CoE achieves higher performance, narrow performance ranges, and strong results even under the worst configuration within each scheme, supporting its robustness across the evaluated configurations. In practice, contrastive sampling (Section 3.1) retains revision as one update path while reconstructing an alternative directly from experience, providing a way to move beyond limitations inherited from the initial skill. Contrast-guided recombination (Section 3.3) then uses preference differences to identify behaviors worth incorporating or preserving. These mechanisms allow updates to address gaps in the initial knowledge while retaining effective strategies.

5 Related Work

Continual learning through external knowledge. LLM agents can learn from interaction by updating external knowledge while keeping model parameters frozen. Reflexion (Shinn et al., 2023) stores verbal reflections to guide subsequent attempts, while ExpeL (Zhao et al., 2024) extracts reusable insights from experience. Beyond reflective memory, Voyager (Wang et al., 2023) accumulates executable skills, and Agent Workflow Memory (Wang et al., 2025) induces reusable workflows. More recent methods improve how experience is organized and updated (Chhikara et al., 2025; Suzgun et al., 2026). ACE (Zhang et al., 2026) maintains evolving playbooks through structured incremental updates, while ReasoningBank (Ouyang et al., 2026) distills generalizable strategies from successful and failed trajectories. Skill-Pro (Mi et al., 2026) and SkillOpt (Yang et al., 2026) further treat reusable skills as explicit optimization targets. Despite these advances, existing methods do not ensure robust learning across different initial skills, as our experiments demonstrate. We study this sensitivity and explore alternative update directions to broaden the behavioral evidence available for skill evolution.

Policy and preference optimization. Another line of work improves agent behavior by updating model parameters from reward signals or preference feedback (Christiano et al., 2017; Meng et al., 2024). Proximal Policy Optimization (PPO) (Schulman et al., 2017) uses a clipped surrogate objective to constrain large policy updates and has been widely adopted for language-model alignment through reinforcement learning from human feedback (Stiennon et al., 2020; Ouyang et al., 2022). Direct Preference Optimization (DPO) (Rafailov et al., 2023) instead learns directly from preference pairs through a reference-relative objective, avoiding both a separately trained reward model and the reinforcement learning stage. These optimization principles have also inspired external skill learning. Skill-Pro (Mi et al., 2026) combines semantic candidate generation with a PPO-inspired verification mechanism. Our approach draws on DPO's pairwise, fixed-reference formulation, but optimizes discrete skill text with a frozen executor, making its gradient-based parameter updates inapplicable. We instead evaluate recorded actions under candidate skills relative to the current skill and use pairwise loss differences to guide targeted textual recombination.

6 Conclusion

We presented Skill-CoE, a framework for robust continual skill evolution in LLM agents with frozen model parameters. By combining skill revision with independent reconstruction and guiding their recombination through within-task trajectory preferences, Skill-CoE explores alternative update directions while preserving effective behaviors. Experiments on ALFWorld, AppWorld, and ScienceWorld show consistent improvements when learning without pre-existing skills and robust learning gains across different initial skills. Ablations and process analyses demonstrate the complementary roles of broader success coverage, paired feedback, and contrast-guided recombination. These results establish external skill evolution as a promising approach to continual agent learning. Future work may extend this framework to longer learning horizons and diverse interactive environments.

GenAI Usage Disclosure

The authors used Generative AI (GenAI) tools solely for the purpose of correcting minor grammatical errors in the manuscript. No part of the research process—including problem formulation, data analysis, model design, coding, experimentation, or interpretation of results—was conducted using GenAI tools. In addition, no GenAI assistance was used in the actual writing or generation of content for this paper, aside from grammar refinement.

Ethics Statement

This work studies continual skill evolution for LLM agents through external procedural knowledge, without modifying the underlying model parameters. Our experiments are conducted on established interactive benchmarks and do not involve sensitive attributes or personally identifiable information. While improved agent learning may inherit risks associated with the underlying LLMs and deployment environments, our work does not introduce mechanisms specifically intended for harmful applications. We follow the ICLR Code of Ethics throughout this work.

Reproducibility Statement

We provide detailed descriptions of the Skill-CoE framework, complete update algorithm, and experimental protocol to support reproducibility. We also provide the complete implementation code at `https://anonymous.4open.science/r/Skill-CoE-2737`. The appendices report implementation details, skill construction and initialization procedures, task splits, ablation variants, and other experimental settings. Together, these resources facilitate reproduction of the reported results and further research.

References

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready AI agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*, 2025. URL https://arxiv.org/abs/2504.19413.

Paul F. Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, volume 30, 2017. URL https://arxiv.org/abs/1706.03741.

Kexin Ding, Yang Zhou, Can Jin, Feng Tong, Mu Zhou, and Dimitris N. Metaxas. Agent skill evaluation and evolution: Frameworks and benchmarks. *arXiv preprint arXiv:2606.11435*, 2026. URL https://arxiv.org/abs/2606.11435.

Brian Ichter, Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. In *Proceedings of the 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pp. 287–318. PMLR, 2023. URL https://proceedings.mlr.press/v205/ichter23a.html.

Yuxuan Liu, Zhaochen Su, Lingyun Xie, Yuhao Zhang, Qing Zong, Jiahe Guo, Zhongwei Xie, Yiyan Ji, Yauwai Yim, Hongyu Luo, Xiyu Ren, Chenyu Ruan, Haoran Li, and Yangqiu Song. SkillRevise: Improving LLM-authored agent skills via trace-conditioned skill revision. In *Findings of the Association for Computational Linguistics: EMNLP 2026*, 2026. URL https://arxiv.org/abs/2606.01139.

Yuchen Ma, Yue Huang, Han Bao, Haomin Zhuang, Swadheen Shukla, Michel Galley, Xiangliang Zhang, and Stefan Feuerriegel. SkillGen: Verified inference-time agent skill synthesis. *arXiv preprint arXiv:2605.10999*, 2026. URL https://arxiv.org/abs/2605.10999.

Yu Meng, Mengzhou Xia, and Danqi Chen. SimPO: Simple preference optimization with a reference-free reward. In *Advances in Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2405.14734.

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. In *International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2602.01869v3.

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.

Evgenii Nikishin, Max Schwarzer, Pierluca D'Oro, Pierre-Luc Bacon, and Aaron Courville. The primacy bias in deep reinforcement learning. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 16828–16847. PMLR, 2022. URL https://proceedings.mlr.press/v162/nikishin22a.html.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, pp. 27730–27744, 2022. URL https://arxiv.org/abs/2203.02155.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T. Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. ReasoningBank: Scaling agent self-evolving with reasoning memory. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2509.25140.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, 2023. doi: 10.1145/3586183.3606763. URL https://arxiv.org/abs/2304.03442.

Reid Pryzant, Dan Iter, Jerry Li, Yin 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/.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://arxiv.org/abs/2305.18290.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://arxiv.org/abs/2302.04761.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. URL https://arxiv.org/abs/1707.06347.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023. doi: 10.52202/075280-0377. URL https://doi.org/10.52202/075280-0377.

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://arxiv.org/abs/2010.03768.

Nisan Stiennon, Long Ouyang, Jeff Wu, Daniel M. Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul Christiano. Learning to summarize from human feedback. In *Advances in Neural Information Processing Systems*, volume 33, 2020. URL https://arxiv.org/abs/2009.01325.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7080–7106. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.eacl-long.333. URL https://aclanthology.org/2026.eacl-long.333/.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16022–16076. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.850. URL https://aclanthology.org/2024.acl-long.850/.

Chenxi Wang, Zhuoyun Yu, Xin Xie, Wuguannan Yao, Runnan Fang, Shuofei Qiao, Kexin Cao, Guozhou Zheng, Xiang Qi, Peng Zhang, and Shumin Deng. SkillX: Automatically constructing skill knowledge bases for agents. *arXiv preprint arXiv:2604.04804*, 2026. URL https://arxiv.org/abs/2604.04804.

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.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. ScienceWorld: Is your agent smarter than a 5th grader? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 11279–11298. Association for Computational Linguistics, 2022. doi: 10.18653/v1/2022.emnlp-main.775. URL https://aclanthology.org/2022.emnlp-main.775/.

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, 2025. URL https://proceedings.mlr.press/v267/wang25bx.html.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-MEM: Agentic memory for LLM agents. In *Advances in Neural Information Processing Systems*, 2025. URL https://arxiv.org/abs/2502.12110.

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.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2210.03629.

Simon Yu, Gang Li, Weiyan Shi, and Peng Qi. PolySkill: Learning generalizable skills through polymorphic abstraction. *arXiv preprint arXiv:2510.15863*, 2026. URL https://arxiv.org/abs/2510.15863.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Pan Lu, Zhi Huang, Carlos Guestrin, and James Zou. Optimizing generative AI by backpropagating language model feedback. *Nature*, 639:609–616, 2025. doi: 10.1038/s41586-025-08661-4. URL https://www.nature.com/articles/s41586-025-08661-4.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou,

and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2510.04618.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024. doi: 10.1609/aaai.v38i17.29936. URL https://ojs.aaai.org/index.php/AAAI/article/view/29936.

Junhao Zheng, Shengjie Qiu, Chengming Shi, and Qianli Ma. Towards lifelong learning of large language models: A survey. *ACM Computing Surveys*, 57(8):193:1–193:35, 2025. doi: 10.1145/3716629. URL https://doi.org/10.1145/3716629.

A Implementation Details and Complete Update Procedure

A.1 Generation, Records, and Scoring

Candidate generation. Both generation paths use the same task descriptions and skill-output format. Revision compares the existing skill with successful and failed executions to retain useful practices and revise omissions, ambiguities, or inapplicable conditions, steps, and constraints. Reconstruction infers reusable procedures from successes and applicability boundaries or practices to avoid from failures. It receives neither the existing skill text nor direct quotations of it, nor the analysis or output of the revision call. Both paths produce complete skills.

Augmented experience. For each comparison task, the existing record is reused and only the two candidates are executed:

$$\begin{aligned} \tau_x(\omega_k) &= \tau_{k,x}, \qquad x \in \mathcal{X}_k, \\ \tau_x(\omega) &\sim \pi_\theta(\cdot \mid x, \omega), \qquad x \in \mathcal{X}_k, \quad \omega \in \{\omega^{\mathrm{rev}}, \omega^{\mathrm{rec}}\}, \\ \widetilde{\mathcal{D}}_k &= \mathcal{D}_k \cup \big\{(x, \tau_x(\omega), R(\tau_x(\omega))) : x \in \mathcal{X}_k,\ \omega \in \{\omega^{\mathrm{rev}}, \omega^{\mathrm{rec}}\}\big\}. \end{aligned} \tag{A1}$$

Subsequent stages retrieve tasks, trajectories, and returns associated with selected cases from $\widetilde{\mathcal{D}}_k$.

Successful replay. Preference pairs omit tied returns. For example, when all three skills succeed with the same return, no pair is formed even though the existing behavior may be worth preserving. We sample $\mathcal{S}_k$ by task from successful records generated by the existing skill in $\mathcal{D}_k$, using the environment's success criterion. Replay supplies behavioral material for generation and does not enter the preference loss. If $\mathcal{E}_k = \varnothing$, the update returns $\omega_{k+1} = \omega_k$ without entering the search.

Execution records. Candidates use the same task initial states, task descriptions, history-retention procedure, and interaction-step limits as the existing records. The existing skill is not re-executed solely to construct preferences. Each record carries task and source identifiers. Even if different sources produce identical text, the identity of each execution record is retained. Matching execution conditions cannot eliminate randomness from model generation or environment changes.

Probability computation. Equation (5) requires $|\mathcal{I}_\tau| > 0$ and scores only action or code tokens. Scoring replaces only the skill text while keeping the task description and preceding record fixed. If the input exceeds the model's context limit, the same history-retention and truncation procedure is used for all skills. The conditioning context contains neither the current token nor future information. Reference scores for the existing skill can be cached.

Surrogate objective. For a given preference pair, the argument to σ in Equation (6) subtracts the same reference value for every candidate. This does not change the ordering of candidate score differences on that pair, but it affects their aggregation across pairs through the nonlinear loss in Equation (6). Mean action log-probability is not the probability of a complete trajectory, and a fixed recorded history may differ from the history a candidate would actually visit. The objective is therefore not equivalent to standard DPO optimization and does not guarantee a higher actual return.

Reference baseline and loss properties. For fixed, nonempty preference evidence $\mathcal{E}_k$, define the change in the preferred–dispreferred score gap relative to the existing skill as

$$m_e(\omega) = \big[s(\omega; \tau_e^+) - s(\omega; \tau_e^-)\big] - \big[s(\omega_k; \tau_e^+) - s(\omega_k; \tau_e^-)\big].$$

Equation (6) can then be written as $\ell_e(\omega) = \log(1 + \exp[-m_e(\omega)])$. Since $m_e(\omega_k) = 0$ for every pair and the task-balanced weights in Equation (7) sum to one, the reference has a fixed baseline:

$$\ell_e(\omega_k) = \log 2, \qquad \mathcal{L}(\omega_k) = \log 2, \qquad \ell_e(\omega) < \log 2 \iff m_e(\omega) > 0.$$

Thus, a pairwise loss below $\log 2$ indicates stronger relative support for the preferred trajectory than under the existing skill. An aggregate loss below this baseline does not imply improvement on every pair.

Writing $\ell(m) = \log(1 + \exp[-m])$, direct differentiation gives

$$\ell'(m) = -\sigma(-m), \qquad \ell''(m) = \sigma(m)\sigma(-m) > 0 \quad \text{for finite } m.$$

The loss is strictly decreasing and convex in the relative margin: increasing the margin reduces the loss, with diminishing benefit as the margin becomes large and positive. These are properties of the scalar scoring function, not parameter-gradient updates or convexity guarantees for the search over skill documents. They do not establish an improvement in actual task return.

A.2 Sampling and Search Branches

Cases are selected by relative preference scores, not trajectory provenance: the preferred trajectory need not have been generated by the donor. A scoring advantage alone does not establish a transferable rule; behavioral analysis and rule alignment assess the relevant content.

Local complementarity versus aggregate improvement. For $e \in \mathcal{E}_x$, let $q_e = 1/(|\mathcal{X}_k^\Delta|\,|\mathcal{E}_x|)$, so that $d_e(\omega) = q_e[\ell_e(\omega^{\text{base}}) - \ell_e(\omega)]$. Alongside the positive contribution $G(\omega)$ in Equation (8), define the magnitude of the candidate's local disadvantages as

$$H(\omega) = \sum_{e \in \mathcal{E}_k} [-d_e(\omega)]_+.$$

Using $a = [a]_+ - [-a]_+$ and the same weights as in the aggregate loss yields

$$\begin{aligned} G(\omega) - H(\omega) &= \sum_{e \in \mathcal{E}_k} d_e(\omega) \\ &= \sum_{e \in \mathcal{E}_k} q_e[\ell_e(\omega^{\text{base}}) - \ell_e(\omega)] \\ &= \mathcal{L}(\omega^{\text{base}}) - \mathcal{L}(\omega). \end{aligned}$$

Consequently, a candidate can have $G(\omega) > 0$ even when its aggregate loss is no lower than that of the base; in that case $H(\omega) \geq G(\omega)$. Its local strengths are offset by weaknesses elsewhere. This decomposition motivates selecting a donor by its positive contribution rather than requiring it to replace the base as a whole. Improvement cases expose those strengths, while preservation cases identify relative weaknesses to guard against during recombination. Neither G nor this decomposition guarantees that the corresponding behaviors can be transferred successfully through text edits.

Case distributions. For a selected donor with $G(\omega_j^{\text{donor}}) > 0$, improvement and preservation cases are drawn using

$$\begin{aligned} P_j^{\text{improve}}(e) &= \frac{[d_e(\omega_j^{\text{donor}})]_+}{G(\omega_j^{\text{donor}})}, \\ P_j^{\text{keep}}(e) &= \frac{[-d_e(\omega_j^{\text{donor}})]_+}{\sum_{h \in \mathcal{E}_k} [-d_h(\omega_j^{\text{donor}})]_+}. \end{aligned} \tag{A2}$$

The resulting sets are $\mathcal{E}_j^{\text{improve}}$ and $\mathcal{E}_j^{\text{keep}}$. Replay cases $\mathcal{S}_j^{\text{replay}}$ are sampled by task from $\mathcal{S}_k$.

Sampling budgets. We select at most two preference pairs for improvement and at most two for preservation, plus at most two successful replay trajectories. Preference pairs from the same task are analyzed together within their respective categories. Equation (A2) is used for weighted sampling without replacement; the remaining weights are renormalized after each draw. If the budget exceeds the number of positive-weight cases, all available cases are used. If the denominator of the preservation distribution is zero, set $\mathcal{E}_j^{\text{keep}} = \varnothing$. If no successful replay case exists, set $\mathcal{S}_j^{\text{replay}} = \varnothing$. Case sampling does not change the overall score: Equation (7) always uses all fixed preference evidence.

Tie-breaking. If skills tie for the lowest loss as the recombination base, prefer the existing skill and otherwise follow a fixed, predetermined order. If donor scores tie, use a fixed insertion order. A candidate with the same loss as the recombination base does not count as an improvement.

No-donor branch. If the candidate pool has no other skill or all other skills have $G = 0$, no donor is available for selecting complementary cases. In this case, we select cases for improvement in proportion to the recombination base's weighted loss:

$$P^{\text{retry}}(e) = \frac{\ell_e(\omega^{\text{base}})}{|\mathcal{X}_k^{\Delta}|\,|\mathcal{E}_x|\,\mathcal{L}(\omega^{\text{base}})}, \qquad e \in \mathcal{E}_x. \tag{A3}$$

We sample at most two improvement cases from this distribution and at most two successful replay trajectories; the preservation-case set is empty. After behavioral analysis, findings are aligned only with text units in the recombination base. The rule-alignment call omits the donor skill, and suggestions are not attributed to rules in another document. Planning, local editing, and scoring then proceed as in Section 3.3.3. If the denominator in Equation (A3) is zero due to numerical underflow, sampling is uniform across tasks and then uniform within each task.

A.3 Materials and Implementation of Staged Generation

Behavioral analysis processes improvement, preservation, and replay materials separately, using execution records from which skill text and direct quotations of it have been removed. Probability scoring uses the original records. Empty material yields an empty analysis result. Behavioral findings retain case identifiers, applicability conditions, and evidence limitations; terminal success is treated as a task outcome, not causal evidence that a particular rule is effective.

Improvement cases are analyzed by comparing preferred and dispreferred trajectories in terms of action choices, execution order, and condition judgments. Preservation cases and successful replays supply behaviors and conditions to retain. Findings B_j record behavioral patterns and their supporting cases. Pairs from the same task are analyzed together within each category.

To connect these findings to editable rules, each skill is partitioned into numbered, continuous text units. Rule alignment is defined as

$$M_j = U_{\text{align}}(B_j, \omega^{\text{base}}, \omega_j^{\text{donor}}). \tag{A4}$$

M_j retains necessary original text, unit and case identifiers, applicability conditions, and conflicts. It distinguishes rules present in a document from new suggestions inferred from trajectories, and identifies whether the base already contains, lacks, or conflicts with the relevant content. This material supports the plan in Section 3.3.3, which specifies edits, evidence, and preservation conditions; deferred suggestions do not become edits. The editor receives only the numbered base and the plan. In explicit form, local editing and assembly are

$$\begin{aligned} \Delta_j &= U_{\text{edit}}(\omega^{\text{base}}, \mathcal{P}_j), \\ \omega_j^{\text{cand}} &= \text{Apply}(\omega^{\text{base}}, \Delta_j). \end{aligned} \tag{A5}$$

Table 5: Stages of skill recombination and their outputs.

Stage	Input	Output and downstream use
Behavioral analysis	Three types of selected records, analyzed separately	Behavioral findings B_j for rule alignment and planning
Rule alignment	B_j and the numbered recombination base and donor skill	Necessary original text, unit and case identifiers, applicability conditions, and conflict notes M_j for planning
Edit planning	B_j, M_j, and feedback from the previous failed attempt	Proposed changes, supporting evidence, preservation conditions, and deferred suggestions $\mathcal{P}_j$ for the editor
Local editing	Numbered recombination base and $\mathcal{P}_j$	`ADD / REPLACE / DELETE` edit blocks Δ_j for programmatic application
Programmatic assembly	Recombination base and Δ_j	A complete candidate for scoring, or a recorded reason if assembly fails
Candidate scoring	Complete candidate and fixed preference evidence	Overall loss determines acceptance; pairwise losses guide donor selection; edits and loss changes feed back into the next planning attempt.

Each plan contains at most three text-unit edits. The program checks edit format, unit identifiers, and length constraints, and preserves the content of untouched units. These structural constraints make edit locations traceable; the candidate's behavioral effects are evaluated by subsequent scoring and actual execution.

If an attempt fails to produce a complete, scorable document, the candidate pool remains $\mathcal{C}_{j+1} = \mathcal{C}_j$, and the attempt still counts toward the generation budget J. Feedback f_j records the attempted edit and reason for failure. For a scored candidate that does not improve, feedback records the edits and the change in overall loss relative to the recombination base. Because the loss is calculated for the complete skill, this change serves as overall feedback for subsequent planning.

A.4 Update Acceptance

Search acceptance only shows that a candidate has a better preference score on fixed trajectories; it does not guarantee a higher return during actual execution. We therefore compare the search output with the existing skill on a fixed validation task set $\mathcal{V}$ used for update selection. Let $R_x(\omega)$ be the measured return of skill ω on validation task x. The two skills use the same task initial state and execution budget. We first calculate the mean validation return:

$$\widehat{R}_{\mathcal{V}}(\omega) = \frac{1}{|\mathcal{V}|} \sum_{x \in \mathcal{V}} R_x(\omega). \tag{A6}$$

We then determine the skill for the next task batch according to whether the search output strictly improves the mean validation return over the existing skill:

$$\omega_{k+1} = \begin{cases} \omega^{\text{out}}, & \widehat{R}_{\mathcal{V}}(\omega^{\text{out}}) > \widehat{R}_{\mathcal{V}}(\omega_k), \\ \omega_k, & \text{otherwise.} \end{cases} \tag{A7}$$

If the search output is ω_k, we retain it directly. Otherwise, an update is deployed only when the validation mean return increases strictly. If it fails this check, the agent continues to use ω_k and does not use validation failures for further revision. Validation records do not enter the training experience, but the validation result does influence the update through deployment selection. Final generalization performance should therefore be evaluated on an independent test task set.

After obtaining ω_{k+1}, the agent executes the next batch $\mathcal{T}_{k+1}$, forms $\mathcal{D}_{k+1}$, and continues the interaction and skill-update process in Section 2.1.

A.5 Complete Update Algorithm

The complete Skill-CoE update integrates contrastive sampling, preference construction, contrast-guided recombination, and validation-based acceptance. Given the current skill and interaction experience, it explores alternative updates and recombines complementary behaviors using fixed preference evidence. Algorithm 1 summarizes the complete procedure.

Algorithm 1 Skill-CoE: contrast-guided skill evolution

Input: Current skill ω_k, experience $\mathcal{D}_k$, comparison tasks $\mathcal{X}_k$, validation tasks $\mathcal{V}$, and generation budget J.
Output: Updated skill ω_{k+1}.
▷ *Contrastive sampling*
Generate revised skill ω^{rev} and reconstructed skill ω^{rec} from $\mathcal{D}_k$ (Eq. (3)).
▷ *Constructing preference pairs*
Execute the candidates on $\mathcal{X}_k$ and combine their trajectories with existing records.
Construct preference pairs $\mathcal{E}_k$ and collect successful replay material $\mathcal{S}_k$.
if $\mathcal{E}_k = \varnothing$ **then**
    return ω_k.
end if
▷ *Contrast-guided skill recombination*
Initialize $\mathcal{C} \leftarrow \{\omega_k, \omega^{\mathrm{rev}}, \omega^{\mathrm{rec}}\}$ and evaluate its skills on the fixed $\mathcal{E}_k$.
Select $\omega^{\mathrm{base}} \leftarrow \arg\min_{\omega \in \mathcal{C}} \mathcal{L}(\omega)$.
$\omega^{\mathrm{out}} \leftarrow \omega^{\mathrm{base}}$; $f_0 \leftarrow \varnothing$.
for $j = 1, \ldots, J$ **do**
    Select a complementary donor from $\mathcal{C} \setminus \{\omega^{\mathrm{base}}\}$ using Eq. (8), or use the no-donor branch.
    Sample improvement, preservation, and replay cases (Appendix A.2).
    Analyze cases and align rules; form an edit plan $\mathcal{P}_j$ using feedback f_{j-1}.
    Generate ω_j^{cand} by editing the fixed ω^{base} (Eq. (9)), and score it on $\mathcal{E}_k$.
    if generation or scoring fails **then**
        Record failure feedback f_j; **continue**.
    end if
    if $\mathcal{L}(\omega_j^{\mathrm{cand}}) < \mathcal{L}(\omega^{\mathrm{base}})$ **then**
        $\omega^{\mathrm{out}} \leftarrow \omega_j^{\mathrm{cand}}$; **break**.
    end if
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{\omega_j^{\mathrm{cand}}\}$; cache losses and record edit feedback f_j.
end for
▷ *Update acceptance*
return ω_{k+1} obtained by validating ω^{out} against ω_k on $\mathcal{V}$ (Eq. (A7)).

B Experimental Details

B.1 Initial Skill Construction

For nonempty initialization, we reserve source tasks from the official training splits of ALFWorld and AppWorld. These tasks are used only to construct initial skill libraries and do not overlap with subsequent training, validation, or test tasks.

Source task groups. In ALFWorld, we assign difficulty labels using expert reference action lengths within each task type together with LLM annotations. We form Easy, Medium, and Hard groups, each containing 12 tasks with the same task-type composition. In AppWorld, we use the official difficulty labels to form three corresponding groups of eight tasks each. These labels describe the source tasks used to construct a library.

Skill extraction. The No-Skill agent executes each source group. We analyze successful and failed trajectories separately, then aggregate the extracted information into an initial library. Each skill contains a name, initiation conditions, an execution strategy, and termination conditions. All groups use the same execution configuration, extraction prompts, and knowledge budget: each library contains one to six skills and at most 1,800 tokens. Empty initialization supplies no preconstructed task-related skills.

Additional initializations. We also construct source groups by task type or application composition and by random partitioning into equally sized groups. These additional initializations examine whether observations about sensitivity depend on grouping source tasks by difficulty.

Table 6: Task counts for continual learning and evaluation. AppWorld uses the test-normal split. Source tasks for initial skill construction are excluded.

Benchmark	Training	Per batch	Validation	Test
ALFWorld	60	12	60	134
AppWorld	45	9	57	168
ScienceWorld	60	12	60	271

Table 7: Success coverage by training batch on ALFWorld. Each batch contains 12 tasks. "Candidates" denotes the two generated skills; "+ parent" also includes the pre-update skill.

	Ours		Double revision	
Batch	Candidates	+ parent	Candidates	+ parent
1	12	12	12	12
2	12	12	12	12
3	12	12	12	12
4	11	12	12	12
5	12	12	7	9

B.2 Task Splits and Learning Protocol

Table 6 summarizes the tasks used after initialization. Each run has five training batches. Source tasks are excluded from these counts and never reused for subsequent training. ALFWorld and AppWorld use identical subsequent task sets, batch partitions, and initial execution orders across initialization conditions. All methods freeze their external knowledge during test evaluation.

For ScienceWorld, we randomly select two instances per task type from each of the official training and validation splits, yielding 60 tasks in each set. The training set is divided into five batches of 12 tasks. From the official test split, we select up to ten instances per type in instance-ID order, yielding 271 test tasks. ALFWorld and ScienceWorld use text environment actions, whereas AppWorld uses code execution and API calls.

B.3 Candidate Execution Statistics

The candidate analysis uses two ALFWorld runs with identical training tasks, batch order, and environment seeds, with no infrastructure failures. Counts follow each task's batch-specific skill versions. They describe candidate executions used during learning and should not be merged with ordinary learning rollouts or final test evaluations. For Ours, the revised and reconstructed candidates succeed on 58 and 56 tasks, respectively, with three revision-only successes and one reconstruction-only success. For double revision, each candidate succeeds on 52 tasks, with three exclusive successes for each candidate.

The methods evolve different parent skills and collect different experience, so the comparison describes their learning runs rather than isolating generation on identical experience. The coverage difference is concentrated in the fifth batch. Table 7 reports coverage by batch. The two-candidate union covers 59 tasks for Ours and 55 for double revision; adding the pre-update parent raises these counts to 60 and 57. Coverage only records whether one of the already executed skills succeeds. It neither specifies a deployable selection rule nor establishes that recombination achieves the same success rate.

B.4 Preference-Pair Improvement Measures

Within batch k, all generated candidates are evaluated on its fixed preference evidence $\mathcal{E}_k$. Restoring the batch index to the notation in Equation (9), write $\omega_{k,j}^{\mathrm{cand}}$ for the candidate actually generated at attempt j, and $\omega_{k,0}^{\mathrm{cand}}$ for the fixed recombination base. For $e \in \mathcal{E}_x$, the task-balanced weight is $q_{k,e} = 1/(|\mathcal{X}_k^{\Delta}|\,|\mathcal{E}_x|)$, matching Equation (7). The weighted fraction of pairs improving over the

preceding candidate is

$$R_{k,j} = \sum_{e \in \mathcal{E}_k} q_{k,e} \, \mathbf{1}\left[\ell_e(\omega_{k,j}^{\text{cand}}) < \ell_e(\omega_{k,j-1}^{\text{cand}})\right]. \tag{B1}$$

Let $\mathcal{K}_3$ denote the batches with at least three generated candidates for the method being analyzed. Table 4 reports $100\overline{R}$, where $\overline{R} = |\mathcal{K}_3|^{-1} \sum_{k \in \mathcal{K}_3} (R_{k,2} + R_{k,3})/2$, and $100\overline{S}$, where

$$\overline{S} = \frac{1}{|\mathcal{K}_3|} \sum_{k \in \mathcal{K}_3} \sum_{e \in \mathcal{E}_k} q_{k,e} \, \mathbf{1}\left[\ell_e(\omega_{k,3}^{\text{cand}}) < \ell_e(\omega_{k,2}^{\text{cand}}) < \ell_e(\omega_{k,1}^{\text{cand}})\right]. \tag{B2}$$

Thus, the second measure concerns two consecutive improvements across three candidates; it does not additionally require $\omega_{k,1}^{\text{cand}}$ to improve over the base. Rejected and repeated candidates remain in sequence, ties do not count as improvements, and no running-best replacement is applied. Batches receive equal weight after task balancing within each batch. These are descriptive statistics conditional on reaching three attempts. The eligible batches and preference evidence need not coincide across methods, and these rates measure the frequency rather than the magnitude of loss reductions.

B.5 Ablation Variants

All variants start from empty initialization and follow the same five-batch training protocol as Ours. Double revision generates both updates by revising the current skill, replacing the independent reconstruction path. The variant without within-task preferences retains the execution outcomes but uses individual trajectory signals instead of winner–loser pairs. Without recombination, the method executes and scores the two updates and selects between them without generating a recombined skill. Random recombination still generates a new skill but randomly selects the base, donor, and improvement/preservation cases. This last comparison evaluates their joint selection strategy.

C Prompt Templates

We report the method-specific prompts for alternative update exploration and contrast-guided recombination. Braced fields denote runtime inputs, and separately invoked stages are marked explicitly. The recombination templates below describe the donor-based path in Section 3.3; sampling, probability scoring, document assembly, and validation acceptance are performed by the implementation, not by these prompts.

Prompt Template for Revision and Independent Reconstruction

Revision channel — one generation call

System instruction: Write reusable procedural knowledge. Return only the complete document as plain Markdown text.

Inputs: Current skill document: {`current_skill`}; successful and failed execution experience: {`batch_experience`}; document budget: {`max_tokens`}.

Instructions: Revise the existing document using both successful and failed execution evidence. Preserve useful procedures and correct unsupported ones. Organize reusable skills with initiation conditions, procedural steps, and termination conditions. Do not include task IDs or instance-specific answers. Keep the complete document within the supplied token budget.

Response requirements: Output the complete revised document, without a JSON wrapper, an outer code fence, or introductory commentary.

Reconstruction channel — two separate calls

Call 1: Independent trajectory analysis

System instruction: Extract reusable procedures from observed actions and outcomes.

Input: Execution experience: {`batch_experience`}.

Prompt Template for Revision and Independent Reconstruction (continued)

Instructions: For each trajectory separately, explain useful behavior and observed failure. Distinguish observed evidence from proposed remedies. Do not infer or quote a prior skill document.

Response requirements: Return per-trajectory lessons grounded in the observed actions and outcomes.

Call 2: Independent skill synthesis

System instruction: Write reusable procedural knowledge. Return only the complete document as plain Markdown text.

Inputs: Per-trajectory lessons: {`trajectory_lessons`}; document budget: {`max_tokens`}.

Instructions: Build a complete procedural document independently from these lessons. Reorganize the procedure as needed. Use initiation conditions, procedural steps, and termination conditions. Do not include task IDs or instance-specific answers. Keep the document within the supplied token budget.

Response requirements: Output the complete reconstructed document, without a JSON wrapper, an outer code fence, or introductory commentary.

Implementation note. Both channels use the same batch experience. The reconstruction calls receive neither the current skill nor the revised candidate; their trajectory views exclude skill text and direct quotations of it.

Prompt Template for Contrastive Behavioral Analysis

System instruction: Analyze observed behavior; no skill documents are provided.

Inputs:

- Selected role: {`role`}, either `ABSORB` (improvement) or `RETAIN` (preservation).
- Cases for one task: {`cases`}, including case identifiers, task descriptions, preferred and dispreferred trajectories, and outcomes.

Role-specific instruction:

- `ABSORB`: Identify reusable winning behavior to absorb and its applicability.
- `RETAIN`: Identify winning behavior and conditions that must be retained during modification.

Comparison instructions:

1. Compare the supplied winning and losing traces together for this task. Winner/loser labels come from environment outcomes. The editing role comes from a separate probability comparison and does not identify which document produced a trajectory.
2. Identify decisive differences in actions, observations, execution order, and condition judgments. Retain the context needed to explain each finding.
3. Separate observations, hypotheses, and missing evidence. Do not infer causal credit from terminal success alone or invent supported commands.
4. State applicability conditions and evidence limitations. Merge redundant observations; omit routine steps unless they explain a difference or a necessary condition. Avoid a step-by-step retelling and generic advice.

Response requirements: Give concise behavioral differences, applicability conditions, and limitations. Do not assign numeric confidence. Findings are retained with their task and case identifiers for downstream use.

Successful replay — a separate analysis call

System instruction: Extract observed successful procedures.

Input: Successful replay trajectory: {`successful_trace`}.

Instructions: State the successful procedure, its applicability, and evidence limitations. Do not invent hidden steps or generalize a command beyond observed support. Retain decisive actions and observations; merge redundant details without omitting necessary conditions.

Prompt Template for Contrastive Behavioral Analysis (continued)

Response requirements: Return a concise account of the observed successful procedure and the conditions under which it is supported.

Implementation note. Improvement and preservation cases are analyzed separately, grouping cases from the same task within each role. Successful replay is analyzed separately as preservation material. These calls receive neither source skill documents nor the probability-score matrix (Appendix A.3).

Prompt Template for Evidence-to-Rule Alignment

System instruction: Map behavioral evidence to existing procedural rules.

Inputs:

- Improvement findings: {`absorb_analysis`}.
- Preservation findings: {`retain_analysis`}.
- Successful replay findings: {`success_analysis`}.
- Numbered recombination base: {`subject_units`}.
- Numbered donor skill: {`reference_units`}.

Instructions:

1. For each relevant improvement finding, identify the corresponding donor unit IDs and summarize their actual rule, including prerequisites, action order, and termination conditions.
2. Compare the finding with the base units. Determine whether the relevant content is already covered, missing, or conflicting. Do not propose duplicate content when the base already covers it.
3. Distinguish transfer supported by a donor rule from new advice inferred only from trajectories. Explicitly state when no matching donor rule exists. Do not claim that a donor rule caused a successful outcome.
4. Treat preservation findings and successful procedures as constraints on what must remain intact, not as a requirement to rewrite every topic.
5. Keep multiple task requirements conjunctive and ordered. Never turn a required sequence into alternative choices or infer an extra operation from the destination.
6. Merge overlapping findings. Refer to source text by unit ID and retain only the rule details needed for the transfer. Preserve necessary prerequisites, ordering, exceptions, and uncertainty; omit unrelated rules.

Response requirements: Write each distinct finding once as a compact entry containing:

- supporting evidence IDs;
- donor unit IDs and the relevant rule;
- base unit IDs and the coverage gap or conflict;
- proposed transfer and essential protections.

Do not repeat the mapping in a second summary table or reproduce complete skill documents.

Implementation note. Here, `SUBJECT` denotes the fixed recombination base ω^{base}, and `REFERENCE` denotes the selected donor ω_j^{donor}. This donor is distinct from the fixed scoring reference ω_k in Equation (6). Base and donor units use separate `U` and `R` identifiers, respectively.

Prompt Template for Skill Modification Planning

System instruction: Plan procedural knowledge modifications from behavior analyses.

Inputs:

- Behavioral findings: {`absorb_analysis`}, {`retain_analysis`}, and {`success_analysis`}.
- Evidence-to-rule mapping: {`rule_transfer`}.

Prompt Template for Skill Modification Planning (continued)

- Numbered fixed base: {subject_units}.
- Previous rejected attempt and feedback, if available: {previous_attempt}.

Instructions:

1. Reconcile improvement and preservation analyses by task. Merge duplicate advice and resolve conflicts using explicit applicability conditions.
2. Use the rule mapping to choose a focused, coherent change addressing behavior that the base does not yet support. Identify the target unit, intended change, supporting case IDs, and why the evidence supports it.
3. Mark essential edits separately from deferred suggestions. The editor can apply at most three unit operations; choose a plan that fits this budget without silently dropping essential changes.
4. Use previous modifications and preference feedback to explain what to narrow or undo. Probability changes concern historical-action support; they are not actual task failures or evidence of a specific rule's causal effect. Treat explanations of their causes as unverified hypotheses.
5. The supplied base is the only edit starting point. A previous rejected candidate is an alternative, not the current document. Any useful previous addition absent from the base must be explicitly re-added; calling it "preserved" does not add it.
6. Plan against individual unit contents, not just section headings. Replacing a heading leaves its existing child lines untouched. Preserve source conditions, action order, exceptions, and useful unaffected behavior. Do not introduce unobserved commands.

Response requirements: Return a concise modification plan, not a final skill document. Specify essential operations, evidence, applicability conditions, preservation requirements, and deferred suggestions. State uncertainty or unresolved conflicts when they affect the edit. Do not reproduce full skills or trajectories, review unrelated rules, or replace necessary qualifications with vague slogans.

Implementation note. The previous-attempt field is empty on the first attempt. Subsequent plans use feedback about the rejected alternative while retaining the same fixed base, as in Section 3.3.3.

Prompt Template for Local Skill Editing

System instruction: Propose minimal local edits to the addressed base; code assembles the final document.

Inputs: Numbered fixed base: {subject_units}; modification plan: {modification_plan}; assembled-document budget: {max_tokens}.

Instructions:

1. Apply the supplied modification plan while preserving unaffected knowledge. Implement its essential changes against this base only. Rejected previous candidates are not the edit base.
2. Return at most three edit blocks. Address only base unit IDs. Prefer REPLACE or ADD; use DELETE only when the selected evidence supports removing that rule.
3. Preserve useful conditions, exceptions, action order, and conditional requirements. Do not copy the donor wholesale.
4. A replacement affects only the named unit; neighboring child lines remain. Do not duplicate those lines inside a replacement heading.
5. The assembled document must fit the supplied token budget. Return NO_CHANGE if no edits are needed.

Response requirements: Output only complete edit blocks or NO_CHANGE. Do not output JSON, outer code fences, commentary, or the complete skill document. Use the following syntax; replace placeholder unit IDs with actual IDs from the supplied base.

Replace one unit:

Prompt Template for Local Skill Editing (continued)

```
<<<REPLACE unit_id>>>
replacement text
<<<END>>>
```

Insert before a unit (use END to append):

```
<<<ADD unit_id>>>
new text
<<<END>>>
```

Delete one unit (empty body):

```
<<<DELETE unit_id>>>

<<<END>>>
```

Implementation note. The editor receives the numbered base and the plan, without raw trajectories or the donor document. The implementation checks edit syntax, unit IDs, and the document budget, and assembles the candidate while preserving untouched units. Scoring and update acceptance remain separate from text generation (Appendices A.3 and A.4).