SKILLSEG: SELF-EVOLVING AGENT SKILLS VIA FEEDBACK ATTRIBUTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Language agents can improve without updating model parameters by revising an external library of reusable skills. However, when multiple skills are invoked within an episode, its terminal outcome does not specify whether the behavior associated with each skill provides evidence for retaining or revising that skill. We propose SkillSeg, a framework for skill-level feedback attribution over trajectory segments, each consisting of consecutive actions executed under the same active skill. For each segment, SkillSeg compares the likelihood of the observed actions under matched prompts with and without the active skill, then combines this contrast with the episode outcome relative to a historical baseline. The resulting sign routes the segment to positive or negative reflection for the corresponding skill, and a likelihood-ratio gate screens the proposed library update. We evaluate SkillSeg with frozen Qwen3.5-9B agents on ALFWorld, WebShop, and AppWorld. SkillSeg achieves $85.07 \pm 1.49\%$ success on ALFWorld, compared with $79.10 \pm 1.49\%$ for a matched segment-level baseline that directly inherits episode labels. It also achieves $36.51 \pm 2.48\%$ success on AppWorld and a score of 55.48 ± 3.64 with $20.00 \pm 1.00\%$ success on WebShop, attaining the highest means among the completed same-backbone methods across these benchmarks. These results support skill-level feedback attribution as a useful mechanism for guiding reusable skill evolution with frozen language models.

1 INTRODUCTION

Language agents can achieve continuous self-improvement without updating model parameters, simply by maintaining and revising a reusable external library of text-based skills. These skills store procedural knowledge, decision strategies, and task-specific guidance that can be reused across different task cycles and updated based on new execution experiences. This non-parametric skill evolution mechanism offers a viable path for long-term adaptation in language agents with frozen parameters.

However, in long-horizon tasks, an agent may invoke multiple skills sequentially within a single task cycle, generating multiple execution segments corresponding to different skills. The environment typically provides only trajectory-level feedback, such as whether the entire task succeeded or failed. This raises a key attribution problem in skill evolution: how to reasonably assign sparse trajectory-level feedback to each individual skill involved. Simply assigning the same terminal label to all invoked skills ignores the fact that behavioral evidence may differ across segments corresponding to different skills at various stages of the trajectory.

This coarse-grained attribution approach can lead to ambiguous skill updates. A failed trajectory may contain useful skill guidance that was effective before subsequent errors occurred; conversely, a successful trajectory may include unhelpful, fragile, or even detrimental skill guidance whose negative impact was compensated by later actions. If the reflection process modifies all involved skills based solely on the shared trajectory-level label, it may incorrectly revise instructions that should be preserved or retain instructions that should be improved. Therefore, the core challenge is: given a segment already associated with a specific skill, how to use the behavioral evidence within that segment to determine how the corresponding skill should be updated, rather than simply propagating the overall trajectory outcome to all invoked skills.

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.

We propose the SkillSeg framework to address the skill-level attribution problem in multi-skill trajectories. For an execution segment already associated with the currently active skill, SkillSeg compares the likelihood of the same set of recorded actions under matched prompt conditions—with and without the skill instruction—to assess the association between the current skill instruction and the segment's behavior. SkillSeg then combines this local behavioral evidence with the difference between the trajectory outcome and a historical performance baseline to provide segment-level context for reflecting on the corresponding skill. Based on this information, the reflection process generates a local patch for the skill, and the candidate skill library subsequently decides whether to commit the update after likelihood-ratio screening.

We first evaluate the role of skill-level attribution through controlled ablation experiments in the ALFWorld environment. When skill-associated segments directly inherit trajectory-level outcomes, the agent achieves a success rate of $79.10 \pm 1.49\%$; with SkillSeg's segment-level attribution mechanism, the success rate improves to $85.07 \pm 1.49\%$. Full-trajectory feedback and a segment-level variant based only on skill-conditioned likelihood comparison serve as additional baselines to analyze the effects of evidence scope and attribution signals. We further evaluate SkillSeg on WebShop and AppWorld to test its performance across different interactive environments. SkillSeg achieves a reward score of 55.48 ± 3.64 and an exact success rate of $20.00 \pm 1.00\%$ on WebShop, and a success rate of $36.51 \pm 2.48\%$ on AppWorld. These results demonstrate that fine-grained attribution over skill-associated segments enables more effective use of trajectory-level supervision to drive skill evolution in language agents with frozen parameters.

The main contributions of this paper are as follows:

1. We formalize the feedback assignment problem in multi-skill trajectories as a fine-grained attribution problem over skill-associated segments, distinguishing trajectory-level outcomes from local behavioral evidence provided by different skill segments.
2. We propose SkillSeg, which compares the likelihood of recorded actions with and without skill instructions under matched conditions and combines this with trajectory outcomes relative to historical performance to provide segment-level attribution information for reflecting on and updating the corresponding skill.
3. We evaluate SkillSeg on three benchmarks—ALFWorld, WebShop, and AppWorld—and conduct controlled ablation studies to examine the impact of evidence scope and skill-level attribution signals on skill evolution performance.

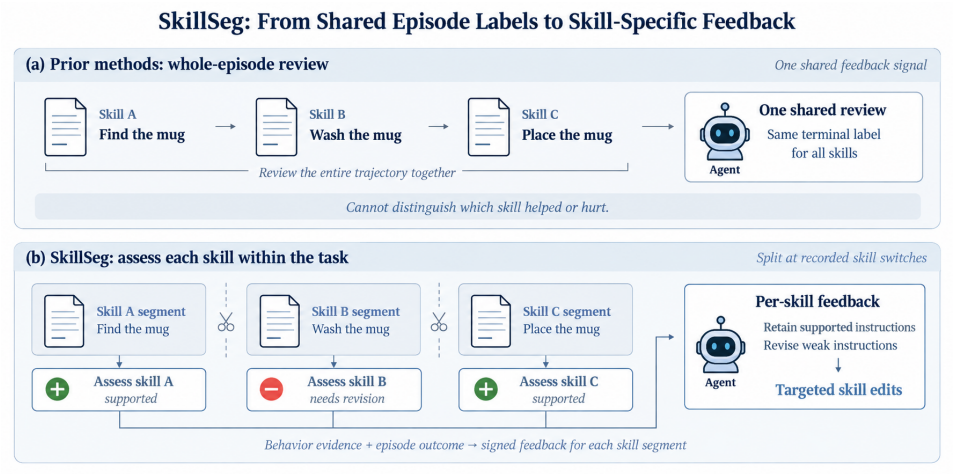

Figure 1: Skill-level feedback attribution from shared episode labels. (a) The whole-episode baseline assigns the same terminal label to all invoked skills. (b) SkillSeg combines behavioral evidence from skill-associated segments with the episode outcome to provide differentiated feedback for each skill. Positive and negative evidence guide targeted edits that retain supported instructions and revise weak ones.

2 Problem Formulation: Feedback for Multi-Skill Trajectories

Let $\Omega = \{\omega_k\}_{k=1}^{K}$ be a library of reusable textual skills and π_θ a frozen language-model actor. At each step, the actor generates an action from the interaction history and the active skill. Skill evolution updates Ω while keeping θ fixed. An episode produces a trajectory

$$\tau = \{(h_t, k_t, a_t)\}_{t=1}^{T},$$

where h_t, k_t, and a_t denote the interaction history, active skill, and recorded action, respectively. The environment supplies a terminal outcome $R_\tau \in \{0, 1\}$.

A **skill-associated segment** I_i is a maximal contiguous interval with the same active skill k_i:

$$I_i = \{t_s, \ldots, t_e\}, \qquad k_t = k_i \quad \forall t \in I_i.$$

A skill switch starts a new segment. Segment boundaries and active-skill identities are available before feedback construction.

An episode-label baseline assigns $f_{\text{episode}}(I_i) = R_\tau$, giving every segment in an episode the same label. This rule does not distinguish the evidence that different skill executions provide for updating their associated skills.

We study **skill-level attribution**: given a segment I_i associated with skill ω_{k_i}, how should its behavioral evidence guide that skill's update? Our goal is to construct feedback indicating which evidence supports preserving the skill's current instructions and which supports revising them. The next section introduces SkillSeg's feedback construction and update procedure.

3 SkillSeg: Skill-Level Feedback Attribution

SkillSeg constructs skill-level attribution signals to guide updates to an external skill library. Given the skill-associated segments defined in Section 2, it combines local action-likelihood evidence with episode outcomes, then organizes that evidence for skill-specific reflection. Candidate updates undergo a separate screening step before entering the library. Figure 2 summarizes the process; both the actor and reflection analyst remain frozen.

3.1 Skill-Conditioned Likelihood Contrast

Consider a segment I_i containing consecutive actions executed under the same active skill k_i. For each step $t \in I_i$, let y_t be the serialized tokens of the recorded action. We construct matched prompts P_t^+ and P_t^- that share the task, interaction history, and system instructions, with the active skill's instructions included only in P_t^+. The action log-likelihoods and segment contrast are

$$\ell_t^{\pm} = \sum_{j=1}^{|y_t|} \log p_\theta(y_{t,j} \mid P_t^{\pm}, y_{t,<j}), \qquad C_i = \frac{1}{|I_i|} \sum_{t \in I_i} (\ell_t^+ - \ell_t^-). \tag{1}$$

Thus, $C_i > 0$ indicates that the skill increases the likelihood of the recorded behavior, whereas $C_i < 0$ indicates that it decreases this likelihood. The contrast measures whether the skill supports the observed actions; the episode outcome provides context for interpreting that support.

3.2 Outcome-Conditioned Segment Score

We compare the terminal outcome R_τ with a historical baseline b_i and combine the resulting advantage with C_i:

$$A_i = R_\tau - b_i, \qquad S_i = C_i A_i. \tag{2}$$

The baseline estimates historical success using the most specific skill and task context with sufficient observations. It is computed from history available before the current rollout batch; Appendix A specifies the estimation and fallback rules.

The score combines two complementary signals: whether the skill supports the recorded behavior and whether the episode outcome exceeds historical performance. Matching signs yield positive feedback; opposing signs yield negative feedback. Table 1 summarizes how these cases inform the associated skill's revision.

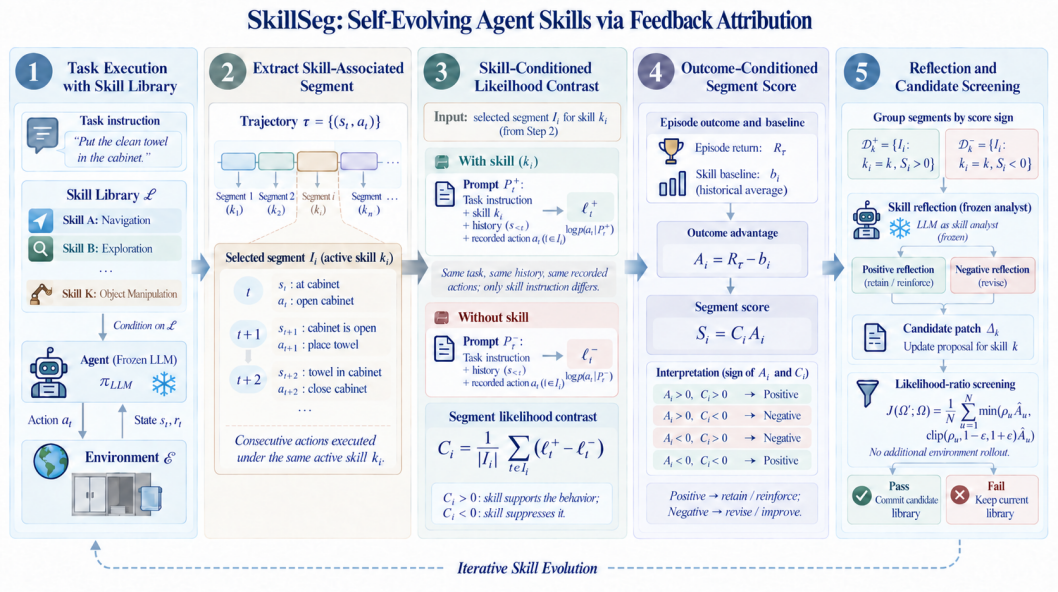

Figure 2: SkillSeg pipeline for skill-level feedback attribution. A frozen actor executes tasks with a skill library, producing skill-associated segments. Matched with-skill and without-skill likelihoods yield C_i, which combines with the outcome advantage $A_i = R_\tau - b_i$ to form $S_i = C_i A_i$. Segments are grouped by skill and score sign for positive or negative reflection. The resulting patches undergo likelihood-ratio screening before accepted updates enter the active library.

Table 1: Feedback assignment from the segment score $S_i = C_i A_i$.

Outcome context	Skill's likelihood effect	Feedback	Interpretation
$A_i > 0$	$C_i > 0$	Positive	Supports behavior in an above-baseline episode
$A_i > 0$	$C_i < 0$	Negative	Suppresses behavior in an above-baseline episode
$A_i < 0$	$C_i > 0$	Negative	Supports behavior in a below-baseline episode
$A_i < 0$	$C_i < 0$	Positive	Suppresses behavior in a below-baseline episode

3.3 Polarity-Conditioned Reflection

We group segments by their associated skill and feedback sign:

$$\mathcal{D}_k^+ = \{I_i : k_i = k,\ S_i > 0\}, \qquad \mathcal{D}_k^- = \{I_i : k_i = k,\ S_i < 0\}. \tag{3}$$

Zero-score segments and those without a usable baseline are excluded from both groups. Each group supplies the frozen analyst with the current skill, recorded actions, environment feedback, and scores C_i, A_i, and S_i.

Positive reflection identifies instructions to preserve or reinforce, while negative reflection identifies instructions to revise. The proposed edits are consolidated into a bounded patch for the corresponding skill body, leaving its termination condition unchanged. The analyst may propose no edit when the evidence is insufficient. Applying the patches produces a candidate library Ω'; reflection prompts and editing constraints appear in Appendices D and A, respectively.

3.4 Candidate Screening

We adopt the clipped likelihood-ratio gate from Skill-Pro (Mi et al., 2026) to screen candidate updates. For each historical decision u, let ℓ_u and ℓ'_u denote the recorded action's log-likelihoods

Table 2: Benchmark results. Unless noted, methods use a Qwen3.5-9B actor. ALFWorld SR is mean ± sample std over three independent runs on 134 unseen tasks. WebShop is averaged over 100 episodes per setting, with mean ± sample std over seeds 42–44 for completed entries. AppWorld uses 168 test-normal tasks with the same three-seed reporting convention where available. SR denotes success rate (%); WebShop score is reward scaled to $[0, 100]$.

	ALFWorld	WebShop		AppWorld
Method	SR ↑	Score ↑	SR ↑	SR ↑
No skill	41.54 ± 0.43	53.67 ± 1.71	18.67 ± 1.15	26.79 ± 1.57
ReAct	63.18 ± 2.40	54.03 ± 2.08	18.33 ± 0.58	26.59 ± 1.50
GEPA	$\underline{79.35 \pm 3.02}$	$\underline{55.02 \pm 2.49}$	$\underline{19.00 \pm 2.00}$	27.38 ± 1.57
Trace2Skill	65.92 ± 4.86	39.23 ± 7.31	14.33 ± 5.69	25.40 ± 0.34
SkillOpt	75.87 ± 0.43	46.91 ± 3.51	18.00 ± 3.00	$\underline{29.17 \pm 2.73}$
Skill-Pro	66.17 ± 1.72	43.79 ± 0.96	17.33 ± 2.08	25.40 ± 2.68
SkillSeg (ours)	$\mathbf{85.07 \pm 1.49}$	$\mathbf{55.48 \pm 3.64}$	$\mathbf{20.00 \pm 1.00}$	$\mathbf{36.51 \pm 2.48}$

under the current and candidate libraries. We compute the stabilized ratio $\rho_u = \exp(\mathrm{clip}(\ell'_u - \ell_u, -10, 10))$ and the historical advantage $\widehat{A}_u = r_u - b_u$, where r_u is the episode outcome and b_u its baseline. The gate score is

$$J(\Omega'; \Omega) = \frac{1}{N} \sum_{u=1}^{N} \min\Big(\rho_u \widehat{A}_u, \mathrm{clip}(\rho_u, 1-\epsilon, 1+\epsilon)\widehat{A}_u\Big), \tag{4}$$

where N is the number of replayed decisions. We use $\epsilon = 0.2$ and accept the entire candidate library when J is finite and exceeds 0.001; otherwise, the current library remains unchanged.

The two scores serve distinct stages: S_i assigns evidence for skill-specific reflection, while J screens the resulting edits using historical actions. Screening requires no new environment rollouts. Appendix A provides the full update algorithm and replay details.

4 Experiments

We evaluate SkillSeg's skill-level feedback attribution through task performance, controlled feedback ablations, generalization across environments and models, and candidate-score analysis.

4.1 Experimental Setup

Benchmarks and protocol. We evaluate SkillSeg on ALFWorld (Shridhar et al., 2021), WebShop (Yao et al., 2022), and AppWorld (Trivedi et al., 2024), covering household tasks, online shopping, and application use. ALFWorld and AppWorld report task success rates; WebShop reports both reward score and exact success rate. The main experiments use frozen Qwen3.5-9B models as the actor and reflection analyst. Each benchmark starts from six generic skills and uses 200 training rollouts in batches of five, yielding 40 evolution iterations. Adaptation updates only the external skill library. Task splits, decoding settings, and runtime details are provided in Appendix B.

Baselines and reporting. We compare against No skill, ReAct (Yao et al., 2023), GEPA (Agrawal et al., 2026), Trace2Skill (Ni et al., 2026), SkillOpt (Yang et al., 2026), and Skill-Pro (Mi et al., 2026). All main-table methods share the frozen Qwen3.5-9B actor, evaluation conditions, and scorer while retaining their own adaptation procedures. We use seeds 42, 43, and 44 and report the mean and sample standard deviation over three independent runs. Success rates are percentages, and WebShop reward scores are scaled to $[0, 100]$. Final evaluation uses frozen skill libraries; evaluation tasks do not participate in skill updates.

Table 3: Feedback ablations on ALFWorld. Success rates are mean ± sample standard deviation over seeds 42, 43, and 44 on 134 unseen tasks.

Feedback strategy	Evidence unit	Polarity source	Success (%) ↑
Trajectory feedback (full)	Full episode	Terminal success / failure	77.11 ± 6.08
Segment + episode labels (traj)	Skill segment	Copy episode outcome	79.10 ± 1.49
Segment + LLM labels	Skill segment	LLM judge	81.59 ± 4.37
Segment + C only	Skill segment	$\mathrm{sign}(C)$	83.08 ± 2.28
SkillSeg ($S = C \cdot A$)	Skill segment	$\mathrm{sign}(S)$	$\mathbf{85.07 \pm 1.49}$

4.2 Main Results

Table 2 shows that SkillSeg achieves the highest mean performance on all three benchmarks with the same frozen Qwen3.5-9B actor. It leads the success-rate comparisons and the WebShop reward score, although the size of the gain varies across environments.

The largest gains over the strongest competing methods appear on ALFWorld and AppWorld. SkillSeg reaches $85.07 \pm 1.49\%$ success on ALFWorld, exceeding GEPA by 5.72 percentage points. On AppWorld, it achieves $36.51 \pm 2.48\%$, a 7.34-point improvement over SkillOpt. These results support the effectiveness of evolving reusable skills from segment-level feedback across household and application-based tasks.

On WebShop, SkillSeg obtains a score of 55.48 ± 3.64 and a success rate of $20.00 \pm 1.00\%$. The gains over GEPA, the strongest baseline on both metrics, are smaller: 0.46 score points and 1.00 percentage point in success rate. Overall, SkillSeg performs well across the three environments, with clearer mean improvements on ALFWorld and AppWorld. Section 4.3 examines the roles of evidence scope and feedback assignment under a matched evolution protocol.

4.3 What Makes Feedback Attribution Effective?

We compare five feedback strategies under the ALFWorld protocol in Section 4.1 to examine the roles of evidence scope and skill-level attribution. All variants share the actor, seed library, rollout budget, patch applicator, and candidate gate. They differ in the evidence supplied to reflection and the rule used to assign feedback labels. Table 3 summarizes the configurations; implementation details are provided in Appendix C.

Restricting reflection from full trajectories to skill-associated segments increases mean success from 77.11% to 79.10% under the same episode-derived labels. With the segment evidence unit fixed, SkillSeg achieves the highest mean success, exceeding copied episode labels by **5.97 percentage points**. It also outperforms LLM-based labels by **3.48 points** and C-only routing by **1.99 points**. These comparisons support combining localized execution evidence with outcome context for skill-level attribution and subsequent skill updates.

4.4 Generalization Across Benchmarks and Models

We evaluate skill libraries evolved on WebShop and AppWorld in ALFWorld, comparing them with No skill and the initial Base skill library. We also apply the same SkillSeg evolution procedure to Qwen3.5-4B and Qwen3.6-35B-A3B, with each model undergoing skill evolution and evaluation on ALFWorld. Table 4 reports both experiments.

The libraries evolved on WebShop and AppWorld improve mean ALFWorld success over Base skill by **11.69** and **10.69 percentage points**, respectively. These gains suggest that the learned updates capture guidance that remains useful beyond the source environment's tasks and interaction interface. Skill evolution can therefore produce reusable instructions whose value extends to a different benchmark.

The additional model experiments examine whether SkillSeg's evolution procedure applies beyond the Qwen3.5-9B backbone used in the main results. Both Qwen3.5-4B and Qwen3.6-35B-A3B support the same procedure, showing its applicability across model sizes and architectures. Together,

Table 4: Cross-benchmark skill transfer and applicability across models on ALFWorld. Results are mean success rate ± sample standard deviation.

Configuration	ALFWorld Success (%) ↑
Cross-benchmark skill transfer	
No skill	41.54 ± 0.43
Base skill (before evolution)	66.42 ± 0.75
Evolved on WebShop	$\mathbf{78.11 \pm 3.68}$
Evolved on AppWorld	77.11 ± 4.11
SkillSeg with additional models	
Qwen3.5-4B	80.60 ± 0.75
Qwen3.6-35B-A3B	$\mathbf{91.29 \pm 1.88}$

the experiments establish two forms of generalization: **transferring evolved skill libraries across environments and applying the evolution method across model configurations**.

4.5 From Candidate Gating to Segment Feedback: Evaluating the Scores

SkillSeg's segment feedback and candidate gate both combine action-likelihood comparisons with relative advantages. The segment score $S = C \cdot A$ compares recorded actions with and without the active skill to guide reflection. The gate score J compares candidate and current libraries using clipped likelihood ratios and advantages to decide whether to accept an update. We evaluate J against independent rollout outcomes and compare candidate acceptance rules to examine this scoring approach in skill evolution.

Agreement with task-level gains. We analyze candidates from 20 consecutive evolution iterations. At each iteration, we compare the current library with the candidate before screening, covering both accepted and rejected candidates. Both libraries are evaluated on the same held-out tasks with fixed random seeds. Tasks are sampled in strata from the 134 unseen ALFWorld tasks, with an average of 50 episodes per evaluation setting. These rollouts are used only for diagnosis and do not affect candidate selection or library updates. We define the success-rate change as

$$\Delta = \mathrm{SR}_{\mathrm{candidate}} - \mathrm{SR}_{\mathrm{old}}.$$

We record J and obtain a separate judgment from GPT-5.6-Sol on whether the candidate improves on the current library. Directional agreement is the fraction of pairs with matching signs; pairs with a zero or missing signal are excluded from the corresponding comparison.

Table 5: Directional agreement for candidates from 20 consecutive evolution iterations. Each denominator includes pairs with two nonzero, available signals.

Signals compared	Agreeing / valid pairs	Agreement (%)
GPT-5.6-Sol judgment vs. Δ	13 / 15	86.7
J vs. Δ	10 / 14	71.4
GPT-5.6-Sol judgment vs. J	15 / 19	78.9

Table 5 shows that J agrees with the observed success-rate change in most valid pairs and often agrees with GPT-5.6-Sol. Historical action likelihoods combined with relative advantages thus provide useful information about candidate quality. Each comparison uses its own valid pair set.

Replacing the candidate acceptance rule. We keep the actor, seed library, training rollout budget, and three-seed setup fixed and replace only the acceptance rule. The alternative evaluates both libraries on a fixed set of 20 validation tasks and accepts the candidate only when its success rate strictly increases. Ties and decreases retain the current library. These validation tasks are separate from the final test set and are used for screening only in this replacement experiment. The main method continues to use the J gate without validation-based library selection.

Table 6: ALFWorld performance with different candidate acceptance rules. Results are mean success rate and sample standard deviation over seeds 42, 43, and 44.

Candidate acceptance rule	ALFWorld Success (%) ↑
Validation (20 tasks)	83.58 ± 2.24
J gate	$\mathbf{85.07 \pm 1.49}$

The two rules achieve similar final performance, with the J gate obtaining a mean success rate 1.49 percentage points higher (Table 6). Under this setup, historical likelihood replay supports skill evolution with performance close to direct validation-based candidate selection.

Connection to segment feedback. These experiments directly assess the candidate-level score J and provide empirical support for combining likelihood comparisons with relative advantages. The segment score S applies this idea to feedback assignment for localized skill segments. Its role is tested in Section 4.3, where the complete score $S = C \cdot A$ outperforms C-only routing. The candidate-level analysis provides indirect support, while the segment ablation provides direct performance evidence for combining local action likelihoods with outcome context in skill evolution.

5 Analysis and Limitations

5.1 What Does Skill-Level Attribution Change?

SkillSeg addresses **skill-level attribution in multi-skill trajectories**: how execution evidence should inform the revision of each participating skill. Given a segment already associated with a skill, it combines the skill-conditioned likelihood of the recorded behavior with episode-relative outcome context to construct feedback for that skill. Segments sharing the same terminal outcome can therefore support different skill updates. This attribution guides reflection toward preserving useful instructions and revising weak ones, rather than assigning every skill the same episode-level label.

5.2 Limitations

SkillSeg constructs skill-level attribution signals to guide skill updates, rather than to estimate the exact reward contribution of each trajectory segment. Its likelihood comparisons evaluate recorded actions under different skill conditions, providing evidence for reflection without measuring the outcomes of alternative executions. The resulting score should therefore be interpreted as an editing signal for the associated skill, not as a causal estimate of its contribution to task success.

The method assumes that active skills and segment boundaries are available and relies on matched likelihood evaluation and historical observations for baseline estimation. Future work could combine this feedback mechanism with richer local supervision or controlled counterfactual rollouts to better assess individual skill contributions.

6 Related Work

Agent Interaction, Memory, and Tool Use. ReAct interleaves reasoning with environment actions (Yao et al., 2023). Reflexion retains verbal feedback across attempts, ExpeL extracts reusable insights from trajectories, and Self-Refine improves outputs through iterative feedback and revision (Shinn et al., 2023; Zhao et al., 2024; Madaan et al., 2023). Persistent memory systems extend experience reuse: Generative Agents retrieves past experiences, MemGPT manages memory tiers, MemoryBank maintains long-term memories, and Agent Workflow Memory stores reusable workflows (Park et al., 2023; Packer et al., 2023; Zhong et al., 2024; Wang et al., 2025). Complementary work on tool use includes Toolformer, which learns API calls through self-supervised training (Schick et al., 2023).

Textual Prompt and Program Optimization. Prompt optimization treats textual instructions as variables that can be revised using task feedback. APE generates and selects instructions, ProTeGi uses textual feedback to guide prompt revisions, and OPRO searches using previous candidates and their scores (Zhou et al., 2023; Pryzant et al., 2023; Yang et al., 2024). Promptbreeder and EvoPrompt apply evolutionary search to prompts (Fernando et al., 2024; Guo et al., 2024). TextGrad propagates textual feedback through computation graphs (Yuksekgonul et al., 2025). DSPy compiles modular language-model pipelines, while MIPRO jointly optimizes instructions and demonstrations (Khattab et al., 2024; Opsahl-Ong et al., 2024). GEPA combines reflection on execution traces with Pareto-based selection to evolve modular programs (Agrawal et al., 2026).

Reusable Procedural Knowledge and Skill Evolution. Reusable libraries organize experience into knowledge that persists across tasks. Voyager accumulates executable skills, LEGO-Prover builds a library of verified lemmas, and ChemAgent maintains a self-updating library of decomposed tasks and solutions (Wang et al., 2024a;b; Tang et al., 2025). Trace2Skill distills trajectory-grounded lessons into transferable skill documents (Ni et al., 2026). SkillOpt directly optimizes textual skills through rollout reflection, bounded edits, and validation (Yang et al., 2026). These methods develop ways to construct and refine reusable knowledge outside model parameters.

Hierarchical Decision Making and Credit Assignment. Temporally extended behavior and credit assignment have long been studied in reinforcement learning. Option-Critic learns options and their termination conditions, HIRO combines high-level goals with low-level control, and COMA uses a counterfactual baseline for multi-agent credit assignment (Bacon et al., 2017; Nachum et al., 2018; Foerster et al., 2018).

Recent language-agent methods connect attribution more directly to skill revision. MASkills combines skill-conditioned credit assignment with hierarchical aggregation in multi-agent systems (Yao et al., 2026). Skill-Pro performs hindsight attribution over skill-controlled trajectory portions, aggregates semantic gradients, and screens candidate updates (Mi et al., 2026). SkillAdaptor identifies actionable faults in failed trajectories and associates them with skills for targeted revision (Yu et al., 2026).

SkillSeg. SkillSeg focuses on skill-level attribution for guiding skill updates. Given a segment already associated with an active skill, it assesses how the recorded behavior should inform revision of that skill. It compares action likelihoods computed by the frozen actor under matched prompts with and without the skill, combines this evidence with the trajectory outcome relative to historical performance, and uses the resulting signal to organize skill-specific reflection. SkillSeg adopts candidate screening from Skill-Pro; its contribution lies in constructing and routing this likelihood-based attribution feedback before textual skill editing.

7 Conclusion

Language agents often invoke multiple reusable skills within a single episode, while terminal rewards provide supervision only at the trajectory level. Consequently, skill executions within the same episode may inherit the same success or failure signal even when they warrant different feedback. SkillSeg addresses this skill-level feedback attribution problem using already identified skill-associated segments. It compares the likelihood of recorded actions under matched prompts with and without the active skill, combines this local likelihood effect with the episode outcome relative to historical performance, and routes the resulting signed evidence to positive or negative reflection for the corresponding skill.

Experiments with frozen Qwen3.5-9B agents on ALFWorld, WebShop, and AppWorld show that SkillSeg improves reusable skill evolution across multiple interactive environments. Controlled ALFWorld ablations further indicate that both evidence localization and feedback polarity matter for the resulting skill updates. Overall, the results support skill-level feedback attribution as a practical mechanism for guiding reusable skill revision while keeping the underlying language model fixed. Future work can extend this framework with richer local supervision, automatic skill attribution, and controlled counterfactual evaluation of individual skill contributions.

Reproducibility Statement

We specify the skill library, frozen actor and reflection models, benchmark splits, evolution budgets, random seeds, feedback equations, candidate screening rule, and evaluation metrics in Sections 2–4. The reported results use three runs with seeds 42, 43, and 44. The fixed old-version source and results baseline are maintained separately from the new manuscript draft.

Ethics Statement

This work evaluates language agents in simulated benchmark environments. The experiments measure task performance and skill-library evolution; they do not establish safety, reliability, or suitability for deployment in unrestricted real-world settings.

Statement on the Use of Generative AI

In this work, we used generative AI tools to design and provide feedback on research methodology and experiments, assist with the execution of experiments, and assist with translation of parts of the manuscript. We also used generative AI tools for sourcing and searching for research information, identifying relevant literature, creating and editing software code used in the experiments, and formatting references. We did not use generative AI tools to generate synthetic datasets, develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for mathematical proofs, assist in writing proofs, propose or refine hypotheses, clean or reformat datasets, support qualitative or thematic data analysis, or interpret the reported results. All AI-assisted work was reviewed by the authors. Experimental procedures, software code, outputs, citations, translations, and reported claims were checked and verified by the authors. We take responsibility for the final content of this work, including all text, claims, results, and 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, Alex Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective Prompt Evolution Can Outperform Reinforcement Learning. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/0e9e708b6f48e14fd0ac29e167413f76-Abstract-Conference.html.

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The Option-Critic Architecture. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017. URL https://ojs.aaai.org/index.php/AAAI/article/view/10916.

Chrisantha Fernando, Dylan Sunil Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-Referential Self-Improvement via Prompt Evolution. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pp. 13481–13544, 2024. URL https://proceedings.mlr.press/v235/fernando24a.html.

Jakob N. Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual Multi-Agent Policy Gradients. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018. URL https://ojs.aaai.org/index.php/AAAI/article/view/11794.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. Connecting Large Language Models with Evolutionary Algorithms Yields Powerful Prompt Optimizers. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/9156b0f6dfa9bbd18c79cc459ef5d61c-Abstract-Conference.html.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan A, Saiful Haq, Ashutosh Sharma, Thomas Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. DSPy: Compiling Declarative Language Model Calls into State-of-the-Art Pipelines. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/f1cf02ce09757f57c3b93c0db83181e0-Abstract-Conference.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. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/91edff07232fb1b55a505a9e9f6c0ff3-Abstract-Conference.html.

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 *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2602.01869.

Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. Data-Efficient Hierarchical Reinforcement Learning. In *Advances in Neural Information Processing Systems*, volume 31, 2018. URL https://papers.nips.cc/paper_files/paper/2018/hash/e6384711491713d29bc63fc5eeb5ba4f-Abstract.html.

Jingwei Ni, Yihao Liu, Xinpeng Liu, Yutao Sun, Mengyu Zhou, Pengyu Cheng, Dexin Wang, Erchao Zhao, Xiaoxi Jiang, and Guanjun Jiang. Trace2Skill: Distill Trajectory-Local Lessons into Transferable Agent Skills, 2026. URL https://arxiv.org/abs/2603.25158.

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. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 9340–9366, 2024. doi: 10.18653/v1/2024.emnlp-main.525. URL https://aclanthology.org/2024.emnlp-main.525/.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as Operating Systems, 2023. URL https://arxiv.org/abs/2310.08560.

Joon Sung Park, Joseph O'Brien, Carrie Jun 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*, pp. 1–22, 2023. doi: 10.1145/3586183.3606763. URL https://doi.org/10.1145/3586183.3606763.

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, 2023. doi: 10.18653/v1/2023.emnlp-main.494. URL https://aclanthology.org/2023.emnlp-main.494/.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, 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://papers.nips.cc/paper_files/paper/2023/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html.

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. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

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.

Xiangru Tang, Tianyu Hu, Muyang Ye, Daniel Shao, Xunjian Yin, Siru Ouyang, Wangchunshu Zhou, Pan Lu, Zhuosheng Zhang, Yilun Zhao, Arman Cohan, and Mark Gerstein. ChemAgent: Self-updating Memories in Large Language Models Improves Chemical Reasoning. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/fa7f64b45970e6a7f8824781e7e01501-Abstract-Conference.html.

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, 2024. doi: 10.18653/v1/2024.acl-long.850. URL https://aclanthology.org/2024.acl-long.850/.

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. *Transactions on Machine Learning Research*, 2024a. URL https://voyager.minedojo.org/assets/documents/voyager.pdf.

Haiming Wang, Huajian Xin, Chuanyang Zheng, Zhengying Liu, Qingxing Cao, Yinya Huang, Jing Xiong, Han Shi, Enze Xie, Jian Yin, Zhenguo Li, and Xiaodan Liang. LEGO-Prover: Neural Theorem Proving with Growing Libraries. In *International Conference on Learning Representations*, 2024b. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/85dca46374dc0f27b4bb5f265b3d17f0-Abstract-Conference.html.

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

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large Language Models as Optimizers. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/3339f19c5fcee3ad74502947a32be9e6-Abstract-Conference.html.

Yifan Yang, Ziyang Gong, Weiquan Huang, Qihao Yang, Ziwei Zhou, Zisu Huang, Yan Li, Xuemei Gao, Qi Dai, Bei Liu, Kai Qiu, Yuqing Yang, Dongdong Chen, Xue Yang, and Chong Luo. SkillOpt: Executive Strategy for Self-Evolving Agent Skills, 2026. URL https://arxiv.org/abs/2605.23904.

Huaiyuan Yao, Xiaoou Liu, Charles Fleming, Tianlong Chen, and Hua Wei. MASkills: Continual Skills Optimization for Multi-Agent LLM Systems, 2026. URL https://arxiv.org/abs/2609.02094.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. WebShop: Towards Scalable Real-World Web Interaction with Grounded Language Agents. In *Advances in Neural Information Processing Systems*, volume 35, 2022. URL https://papers.nips.cc/paper_files/paper/2022/hash/82ad13ec01f9fe44c01cb91814fd7b8c-Abstract-Conference.html.

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://openreview.net/forum?id=WE_vluYUL-X.

Zhuoyun Yu, Xin Xie, Wuguannan Yao, Chenxi Wang, Lei Liang, Xiang Qi, and Shumin Deng. SkillAdaptor: Self-Adapting Skills for LLM Agents from Trajectories, 2026. URL https://arxiv.org/abs/2606.01311.

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.

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.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. MemoryBank: Enhancing Large Language Models with Long-Term Memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19724–19731, 2024. doi: 10.1609/aaai.v38i17.29946. URL https://ojs.aaai.org/index.php/AAAI/article/view/29946.

Yongchao Zhou, Andrei Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large Language Models are Human-Level Prompt Engineers. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=92gvk82DE-.

A Implementation Details

This appendix describes how SkillSeg converts already localized skill-associated segments into feedback, constructs candidate skill patches, and decides whether a candidate update enters the next library version. SkillSeg assumes that each rollout already contains the active-skill annotation supplied by the runtime. Its role is to organize the behavioral evidence in an existing segment into an auditable signal for revising the associated skill.

A.1 Rollout and Segment Representation

Each execution receives a unique execution identifier. Repeated executions of the same task therefore remain separate, and different segments within one execution are separately addressable. A rollout stores the task identifier and type, execution and step indices, the observation or input state, the executed action, environment feedback, the active skill identifier, the skill text used at execution time, the episode outcome, and the versions of the skill library and historical baseline. The complete decision input used to generate each action is also retained so that likelihood scoring can reconstruct the original decision context without rerunning the environment.

A skill-associated segment is constructed by scanning the executed actions in temporal order. Consecutive actions with the same active skill identifier form one maximal segment. A switch to another skill closes the current segment and starts a new one. Actions without an active skill are marked as unassigned, close any open segment, and do not contribute to a skill's segment score. For example, the sequence skill_1, skill_1, skill_1, NONE, skill_2, skill_2 produces a three-step segment for skill_1, one unassigned action, and a two-step segment for skill_2. Segment boundaries are determined only by the recorded active-skill annotations; reward, likelihood, and textual similarity are not used to recluster the trajectory.

For auditing and replay, each segment stores its segment identifier, trajectory and execution identifiers, associated skill, start and end positions, action count, episode outcome, baseline metadata, library version, and action-level scoring evidence. Keeping execution, trajectory, and segment identifiers distinct prevents repeated runs of the same task from being conflated with different skill segments in one run.

A.2 Matched Prompt Construction and Likelihood Scoring

For each recorded action in a segment, SkillSeg constructs two matched decision inputs. The with-skill input contains the fixed decision instructions, the recorded observation, and the skill text that was active during execution. The without-skill input uses the same system instructions, observation, action format, target serialization, tokenizer, and scoring mode, but replaces the active skill content with an empty skill wrapper. Thus, the active-skill condition is the only intended difference between the two inputs.

Let y_t denote the serialized action target at step t. SkillSeg scores only the target action tokens and excludes prompt tokens and hidden reasoning tokens. The with-skill and without-skill log-likelihoods are

$$\ell_t^+ = \sum_{j=1}^{|y_t|} \log p_\theta(y_{t,j} \mid P_t^+, y_{t,<j}), \qquad \ell_t^- = \sum_{j=1}^{|y_t|} \log p_\theta(y_{t,j} \mid P_t^-, y_{t,<j}). \tag{5}$$

The step-level contrast is $C_t = \ell_t^+ - \ell_t^-$. For a segment I_i, SkillSeg averages the contrasts of its valid actions,

$$C_i = \frac{1}{|I_i|} \sum_{t \in I_i} C_t. \tag{6}$$

A positive C_i means that conditioning on the active skill raises the likelihood of the recorded actions on average; a negative value means that it lowers their likelihood.

A.3 Historical Baseline and Segment Attribution

Each segment inherits the outcome of its containing episode. For binary-success environments, this outcome is the hard episode reward. For environments configured with a scalar task score, the

corresponding episode-level reward is used on its evaluation scale. Let this outcome be R_τ. The baseline is computed from the historical rollout buffer using the most specific context with sufficient observations. The implementation falls back in the following order: skill–task, skill, task type, and global history. The default minimum counts are five observations for the first three levels and ten observations for the global level. If no level has enough observations, the baseline is marked unavailable.

To prevent information leakage, the baseline state saved before the current batch is used to score every segment in that batch. Only after all current segments have been scored are the new episode outcomes inserted into the historical state. Duplicate executions are removed using their execution identifiers before the updated state is saved. Consequently, the current rollout does not serve as its own baseline, and the score of one example cannot change merely because another example in the same batch was processed earlier.

For segment i, the outcome-relative advantage and signed segment score are

$$A_i = R_{\tau(i)} - b_i, \qquad S_i = C_i A_i, \tag{7}$$

where b_i is the selected historical baseline. The sign of S_i determines the reflection route: $S_i > 0$ is positive evidence and $S_i < 0$ is negative evidence. A score of zero is neutral and is not used to support either polarity. Neutral and unavailable segments are kept in the audit record but are excluded from positive and negative patch generation. A segment is unavailable when its baseline cannot be estimated, its input or action target is missing, or likelihood scoring fails.

Each segment annotation records C_i, $R_{\tau(i)}$, the selected baseline, A_i, S_i, the baseline source and count, an availability flag, and the number of skipped actions or states. These annotations make it possible to reconstruct why a segment was routed to a particular reflection branch.

A.4 Reflection and Patch Construction

After scoring, evidence is grouped by the associated skill identifier and separated by polarity. For each skill, the reflection input contains the current skill document, the task context, the local segment, the observed actions and environment feedback, and the segment annotation. Positive and negative evidence are presented separately so that the editor can identify which parts of the current instructions are supported and which parts require revision. The editor may return an empty patch when the available evidence does not justify an edit.

A proposed patch targets only the skill that produced the corresponding segment. Each edit must cite an existing segment identifier and may use only non-neutral, available evidence belonging to that skill. The patch is restricted to bounded body-level additions or revisions; it cannot change the skill identifier, skill description, initiation condition, termination condition, routing policy, another skill, or a library-level policy. In the reported implementation, a skill receives at most four body-level additions in one update, and an empty patch is valid when reflection finds no supported change. Patch validation rejects empty instructions, unknown evidence identifiers, evidence from another skill, neutral or unavailable evidence, and edits outside the permitted body-level format.

A.5 Candidate Screening, Library Commit, and Worked Example

A valid patch first produces a candidate library separate from the active library. The candidate is rejected immediately if no edit is applied or if the old and candidate libraries generate identical decision inputs for the affected skills. Otherwise, the screener replays historical transitions for the affected skills. Each replay transition contains the skill identifier, input state, recorded action, episode outcome, baseline, advantage, and task type. When the replay buffer is larger than the screening budget, the implementation uses a deterministic head–tail sample with a maximum of 64 transitions and requires at least one valid transition.

For every replay transition, the old and candidate libraries score the same action. Let ℓ_u and ℓ'_u denote the corresponding old and candidate log-likelihoods. The stabilized likelihood ratio and its clipped version are

$$\rho_u = \exp(\mathrm{clip}(\ell'_u - \ell_u, -10, 10)), \qquad \widetilde{\rho}_u = \mathrm{clip}(\rho_u, 1-\epsilon, 1+\epsilon), \tag{8}$$

Algorithm 1 One iteration of SkillSeg

Require: Library Ω, segment history H, gate history G, batch size B
Collect B episodes under Ω with frozen models
Extract skill-associated segments and read baselines from H
for each segment I_i with sufficient history **do**
    Compute C_i, A_i, and S_i using Equations (1)–(2)
    Route $S_i > 0$ to $\mathcal{D}^+_{k_i}$ and $S_i < 0$ to $\mathcal{D}^-_{k_i}$; omit zero scores
end for
Update H with the collected batch
Reflect on each nonempty skill/polarity group and form candidate Ω'
if a usable candidate exists **then**
    Compute $J(\Omega'; \Omega)$ on historical actions from G
    if J is finite and $J > 0.001$ **then**
        $\Omega \leftarrow \Omega'$
    end if
end if
Advance the gate history G
return Ω, H, G

with $\epsilon = 0.2$. The candidate screening score is

$$J(\Omega'; \Omega) = \frac{1}{N} \sum_{u=1}^{N} \min \left(\rho_u \widehat{A}_u, \widetilde{\rho}_u \widehat{A}_u \right), \tag{9}$$

where N is the number of replayed decisions and $\widehat{A}_u = r_u - b_u$ is the historical advantage defined in Section 3.4. A candidate is accepted when J is finite and exceeds the default acceptance margin of 0.001. It is rejected when no edit is applied, the candidate does not change the relevant decision input, the replay set is unusable, an advantage is unavailable, J is non-finite, or J does not exceed the margin. Candidate screening uses saved replay transitions and does not rerun the environment; independent validation rollouts, when used for analysis, measure the task-level effect of an accepted candidate without changing the definitions of C_i, A_i, or S_i.

When the candidate library passes screening, all of its proposed changes are committed together and the entire candidate replaces the active library. If it is rejected, the current library remains unchanged. The implementation records the candidate patch and library, validation results, screening score, replay details, affected skill identifiers, gate decision, and the committed library version regardless of whether the candidate is accepted.

As a concrete illustration, suppose a two-action segment has step-level contrasts $C_3 = 0.7$ and $C_4 = 0.2$. Its segment contrast is $C_i = 0.45$. If the episode outcome is $R_\tau = 1.0$ and the historical baseline is $b_i = 0.6$, then $A_i = 0.4$ and $S_i = 0.18$. The segment is therefore routed to positive reflection for its associated skill. The resulting candidate patch is then evaluated on the replay transitions through Eq. (9); only a candidate whose finite score exceeds the acceptance margin is committed.

The complete update flow is therefore:

$$\text{rollout} \;\rightarrow\; \text{localized segment} \;\rightarrow\; (C_i, A_i, S_i) \;\rightarrow\; \text{reflection patch} \;\rightarrow\; J \;\rightarrow\; \text{library update}.$$

The gate then accepts or rejects the candidate library update.

Algorithm 1 summarizes one evolution iteration.

B Experimental Setup and Run Configuration

B.1 Experimental Goals and Common Protocol

The main experiments evaluate skill-library evolution while keeping the language-model parameters frozen. Model parameters remain fixed during training rollout collection, skill-library updates,

candidate screening, and final evaluation. Adaptation is performed only by modifying the external text skill library.

The main-table methods use a frozen Qwen3.5-9B actor. Unless otherwise stated in Appendix C, methods use the same task partitions, environment interfaces, evaluation metrics, and random seeds. Each method retains its own experience organization, skill-construction procedure, and update rule.

Each benchmark maintains an independent skill library, rollout history, and baseline state. Rollouts, skill versions, and historical statistics are not shared across benchmarks. Evaluation trajectories are not written back to the training history and do not update the skill library.

B.2 Models and Inference Configuration

SkillSeg uses a frozen Qwen3.5-9B model as both the action actor and the reflection analyst. The generation settings are as follows:

Table 7: Generation settings used during training, skill evolution, and final evaluation.

Configuration	Training and evolution	Final evaluation
Model	Qwen3.5-9B	Qwen3.5-9B
Thinking	Enabled	Enabled
Temperature	0.7	0.6
Top-p	0.95	0.95
Maximum generation length	16,384 tokens	16,384 tokens

Skill selection, action generation, and reflection calls use the fixed settings for their corresponding phase. Final evaluation loads the evolved skill library and keeps the model and tokenizer unchanged.

Likelihood scoring for candidate screening uses saved states, actions, and the corresponding skill-library versions. It does not execute the environment again, generate new actions, or change the task state. The matched prompts, target tokens, and scoring rules are described in Appendix A.

B.3 Initial Skill Library and Evolution Budget

SkillSeg starts from six general skills on each benchmark. Runs with different random seeds on the same benchmark use the same initial skill library. Skill text is adapted to the task interface of each benchmark.

Skill evolution proceeds in fixed batches:

- each evolution iteration collects five training rollouts;
- the batch is used to produce feedback, reflection results, and candidate skill patches;
- a candidate library that passes the gate replaces the active library in full;
- a rejected candidate leaves the current skill library unchanged;
- each run uses a total budget of 200 training rollouts;
- the budget gives 40 evolution iterations.

ALFWorld and WebShop each use 200 training episodes. AppWorld uses 200 training rollouts sampled from its official training-task pool. Every rollout receives a unique execution identifier so that repeated executions of the same task remain separate.

Training rollouts are used for skill evolution, historical baselines, and candidate screening. Final test tasks do not participate in skill-library updates or historical statistics.

B.4 Benchmark Task Splits

ALFWorld. ALFWorld uses training tasks for rollout collection and skill-library evolution, and reports final results on 134 unseen evaluation tasks. Evaluation tasks do not participate in skill evolution, baseline updates, or candidate screening.

WebShop. Each WebShop evaluation uses a fixed set of 100 evaluation episodes. We report both the reward score returned by WebShop and the exact success rate. Training uses 200 rollouts organized into 40 iterations of five rollouts. Training rollouts and evaluation episodes are stored separately.

AppWorld. AppWorld uses the official test-normal split of 168 tasks for final evaluation. During training, 200 rollouts are collected from the official pool of 90 training tasks and organized into 40 iterations of five rollouts. Test tasks do not participate in skill evolution, baseline estimation, or candidate screening.

The main benchmark settings are summarized below.

Table 8: Benchmark settings used in the main experiments.

Benchmark	Training source	Training rollouts	Evaluation size	Metrics
ALFWorld	Training task set	200	134 unseen tasks	Success rate
WebShop	Fixed training manifest	200	100 episodes	Score, exact success
AppWorld	90 official training tasks	200	168 test-normal tasks	Success rate

B.5 Random Seeds and Reporting

The main experiments use random seeds 42, 43, and 44. Each seed corresponds to an independent skill-evolution and evaluation run with its own rollout sequence, historical statistics, and skill-library versions. The seeds control task sampling, environment initialization, and request-level random states used by the experiment interface.

Unless stated otherwise, every final metric is reported as the mean and sample standard deviation over the three independent runs. For results m_1, m_2, m_3, the mean is

$$\bar{m} = \frac{1}{3}\sum_{k=1}^{3} m_k, \tag{10}$$

and the sample standard deviation is

$$s_m = \sqrt{\frac{1}{3-1}\sum_{k=1}^{3}(m_k - \bar{m})^2}. \tag{11}$$

We report results as $\bar{m} \pm s_m$. The standard deviation measures variation across independent evolution runs, rather than the task-level standard error within one run.

B.6 Evaluation Metrics

ALFWorld and AppWorld use binary task success rate,

$$\mathrm{SuccessRate} = \frac{\text{number of successfully completed tasks}}{\text{number of evaluation tasks}} \times 100. \tag{12}$$

For WebShop, the reward score is the mean environment reward over evaluation episodes, scaled to $[0, 100]$. The exact success rate is the percentage of episodes that fully satisfy the task requirements.

Because reward definitions differ across benchmarks, WebShop score, ALFWorld success rate, and AppWorld success rate are reported separately and are not combined into one cross-benchmark number.

B.7 Skill-Library Versions, Gate Decisions, and Final Freeze

At every evolution iteration, SkillSeg starts from the current active skill library and produces a candidate library. The gate makes one accept or reject decision for the entire candidate library.

If the candidate passes the gate, all of its proposed skill modifications are committed together, and the candidate replaces the active library for the next iteration. If it is rejected, the entire current library remains unchanged.

This process continues until 40 evolution iterations are completed or the rollout budget is exhausted. The final evaluation uses the active skill library that remains after the last gate decisions. For the main method, this library is the result of repeated gate-filtered updates and is not selected by an additional validation set. The replacement experiment in Section 4.5 instead uses 20 validation tasks, separate from the final test set, to screen candidates.

After final evaluation begins, the skill library is frozen. No reflection, patch generation, candidate screening, or baseline update is performed. Each run stores the model and decoding configuration, benchmark and task manifest, random seed, initial skill library, execution identifiers for each rollout batch, candidate libraries, gate decisions, active library versions after each iteration, the final frozen library, and per-task evaluation outputs.

These records distinguish candidate libraries, rejected versions, libraries actually committed after each gate decision, and the final library used for testing.

C Ablation Configurations and Feedback Protocols

This appendix describes the five feedback strategies in Table 3. We specify the evidence supplied to reflection, the feedback labels, and the shared editing and screening procedure. Results are reported in Section 4.3; runtime settings and prompts are provided in Appendices B and D.

C.1 Shared Protocol and Configuration Overview

All five configurations use a frozen Qwen3.5-9B actor, the same six-skill seed library, and a budget of 200 training rollouts, collected in batches of five over 40 evolution iterations. We use seeds 42, 43, and 44 and evaluate on 134 unseen ALFWorld tasks. The patch applicator, candidate screening rule, and final evaluation protocol are shared. The configurations vary the evidence unit and the source of feedback labels.

Feedback strategy	Evidence unit	Label source
Trajectory feedback (full)	Full episode	Terminal success or failure
Segment + episode labels (traj)	Skill segment	Copied episode outcome
Segment + LLM labels	Skill segment	LLM judge
Segment + C only	Skill segment	$\mathrm{sign}(C_i)$
SkillSeg	Skill segment	$\mathrm{sign}(C_i A_i)$

The shared settings do not require identical execution trajectories. As their skill libraries evolve, the configurations can produce different actions, environment feedback, and subsequent edits.

C.2 Full-Trajectory and Episode-Label Feedback

Trajectory feedback (full) supplies the complete episode transcript to reflection, including the task, actions, and environment feedback. Successful episodes provide positive feedback, and failed episodes provide negative feedback. Reflection can inspect the full trajectory, but each skill-associated segment does not receive a separate feedback label.

Segment + episode labels (traj) restricts the evidence to the consecutive actions and environment feedback associated with the target skill. Each segment inherits its episode outcome: all segments in a successful episode are positive, and all segments in a failed episode are negative.

These two configurations use the same episode-derived labels and differ in evidence scope. The segment-label configuration does not use likelihood contrasts or an LLM judge to reassign feedback.

C.3 LLM-BASED SEGMENT FEEDBACK

Segment + LLM labels uses the same skill-associated segments as the other segment-level configurations. An LLM judge evaluates the target skill using the segment's actions and environment feedback. Any surrounding steps serve as context; the judgment focuses on the target segment. The judge does not receive C_i, A_i, or S_i.

The judge returns one of four labels:

- `helpful`: the focused execution makes useful progress within the skill's responsibility;
- `harmful`: the focused execution contains avoidable mistakes, loops, or poor decisions;
- `mixed`: the segment contains both useful and unfavorable evidence;
- `insufficient`: the available evidence does not support a clear judgment.

The labels `helpful` and `harmful` map to positive and negative feedback, respectively. The labels `mixed` and `insufficient` are treated as neutral and do not support edits. Labeled segments then enter the shared evidence-grouping and editing procedure. Appendix D.5 provides the judge prompt.

C.4 LIKELIHOOD-BASED SEGMENT FEEDBACK

Segment + C only assigns positive feedback when $C_i > 0$ and negative feedback when $C_i < 0$. A zero contrast is neutral. A positive contrast means that including the active skill raises the likelihood of the recorded actions under matched prompts; a negative contrast means that it lowers their likelihood.

This configuration uses the same segment definition, matched prompts, and likelihood scoring as SkillSeg. Its feedback labels do not incorporate the episode outcome relative to historical performance.

SkillSeg combines the likelihood contrast with the outcome context:

$$A_i = R_\tau - b_i, \qquad S_i = C_i A_i.$$

Segments with $S_i > 0$ provide positive evidence, those with $S_i < 0$ provide negative evidence, and those with $S_i = 0$ are neutral. Relative to C-only routing, SkillSeg adds the episode outcome relative to the historical baseline when assigning feedback. Appendix A describes likelihood aggregation and baseline estimation.

C.5 EVIDENCE ORGANIZATION AND SKILL EDITING

The four segment-level configurations use the shared editor prompt in Appendix D.6. They retain the same segment representation and supply labels from their respective feedback rules.

Evidence is grouped by target skill, with positive and negative segments kept distinct. Each segment retains its task information, target skill, segment identifier, step range, actions, and environment feedback. Segment identifiers link proposed edits to their supporting evidence.

The shared editor first produces separate positive and negative reflections, then proposes edits for the target skill. Neutral evidence cannot justify an edit, and evidence from another skill cannot support the target skill's patch. All edits follow the same patch-applicator constraints and leave skill identifiers, routing, and library-level settings unchanged. The editor may return no edit when the evidence does not support a reusable change.

C.6 CANDIDATE SCREENING AND FINAL EVALUATION

All five configurations use the same candidate screening rule. Proposed edits form a candidate library, and likelihood replay on saved historical transitions produces the screening score J. Screening uses recorded states and actions without rerunning the environment.

An accepted candidate replaces the entire active library for the next iteration, committing all proposed modifications together; a rejected candidate leaves the current library unchanged. Each run

records candidate patches, screening scores, gate decisions, and committed skill versions. Appendix A specifies the screening formula and acceptance condition.

After the evolution budget is exhausted, evaluation uses the active library retained after the final gate decisions, without an additional validation set for version selection. The library remains frozen during evaluation, and evaluation tasks do not enter reflection, baseline updates, or candidate screening. All configurations use the same unseen tasks and metrics and report the mean and sample standard deviation over three independent runs, as described in Appendix B.

D Prompt Templates

This appendix provides the ALFWorld prompt templates used for action generation, reflection, and the feedback ablations. Fields in braces are filled with the corresponding skill, task, or segment data.

D.1 Dynamic Fields

The main fields are listed below. The actor prompt also uses `{skill_title}`, `{skill_initiation}`, `{skill_policy}`, and `{skill_termination}` for the active skill's components.

Field	Content
`{skill}`	Current target skill document.
`{task}`	Task description.
`{segment_id}`	Unique segment identifier.
`{focused_steps}`	Actions and environment feedback associated with the target skill.
`{polarity}`	Feedback label assigned to the segment.
`{evidence_segment_ids}`	Identifiers of the segments supporting an edit.
`{input_state}`	State input saved when the action was generated.
`{action}`	Action actually executed.

D.2 Actor and Matched Scoring Prompts

Likelihood scoring uses the following base system prompt.

Base system prompt

```
You are an expert agent operating in the ALFRED Embodied Environment.
```

The with-skill prompt includes the active skill's instructions.

With-skill action prompt

```
[ACTIVE OPTION]
Name: {skill_title}

Initiation:
{skill_initiation}

Policy:
{skill_policy}

Termination:
{skill_termination}

[EXECUTION]
Use this option only while its Initiation matches the current state
and its Termination is not already satisfied.
```

```
Follow the Policy to choose exactly one valid ALFWorld action from the
admissible actions.
Output ONLY one valid environment action wrapped in
<action>...</action>.
```

For the without-skill condition, we remove the active skills instructions from the scoring input while retaining an empty `[ACTIVE OPTION]` block and keeping the remaining context unchanged.

Without-skill action prompt

```
[ACTIVE OPTION]

[EXECUTION]
Use this option only while its Initiation matches the current state
and its Termination is not already satisfied.
Follow the Policy to choose exactly one valid ALFWorld action from the
admissible actions.
Output ONLY one valid environment action wrapped in
<action>...</action>.
```

The two conditions use the same recorded state, action format, and target action, with the skill condition changed as shown above.

D.3 Positive Reflection Prompt

This template processes segments already assigned to the positive (helpful) group for a target skill.

Positive reflection

```
You are an expert success-pattern analyst for multi-skill ALFWorld
agents.

You will be given MULTIPLE skill segments that were already labeled as
positive / helpful for one target skill, plus the current skill
document. Your job is to encode reusable local success patterns for
that target skill only.

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

You will be told the maximum number of edits (the budget L). Produce
AT MOST L edits. You may produce fewer if warranted.

Every non-add edit MUST include `edit_skill` and it must name exactly
one existing skill. Also include `target_field` (default `body`) and
`operation_scope` (`skill`).

## Generalization Requirements

Positive segments are local evidence, not instructions to memorize
their exact path. Only propose edits that capture reusable principles.

A valid success edit should:
- describe a transferable decision rule or phase transition
- avoid concrete object names, receptacle names, room layouts, and
instance numbers unless the rule applies to the entire ALFWorld task
family
```

```
- use generic categories such as target object, source receptacle,
destination receptacle, container, appliance, inventory, and observed
feedback
- state when the rule should stop being applied
- strengthen an existing broadly useful strategy without narrowing it
to one episode

Prefer no edit over an episode-specific edit.

Respond ONLY with a valid JSON object:
{
  "batch_size": <number of segments analysed>,
  "contribution": "helpful",
  "success_patterns": ["<pattern 1>", "<pattern 2>"],
  "patch": {
    "reasoning": "<why these patterns are worth encoding>",
    "edits": [
      {"op": "append", "edit_skill": "<existing skill id>",
      "target_field": "body", "operation_scope": "skill", "content":
      "<markdown>"},
      {"op": "insert_after", "edit_skill": "<existing skill id>",
      "target_field": "body", "operation_scope": "skill", "target":
      "<heading/text>", "content": "<markdown>"},
      {"op": "replace", "edit_skill": "<existing skill id>",
      "target_field": "body", "operation_scope": "skill", "target":
      "<old text>", "content": "<new text>"},
      {"op": "delete", "edit_skill": "<existing skill id>",
      "target_field": "body", "operation_scope": "skill", "target":
      "<exact text to remove>"}
    ]
  }
}
```

D.4 Negative Reflection Prompt

This template processes segments already assigned to the negative (harmful) group for a target skill.

Negative reflection

```
You are an expert failure-analysis agent for multi-skill ALFWorld
agents.

You will be given MULTIPLE skill segments that were already labeled as
negative / harmful for one target skill, plus the current skill
document. Your job is to propose concise body edits that fix reusable
local failure patterns for that target skill only.

## Analysis Process
1. Read every negative segment in the minibatch.
2. Focus on the segment's own steps and environment feedback; treat
any surrounding context as secondary.
3. Identify common, reusable failure patterns across segments of this
skill.
4. Propose skill edits that address those patterns -- not
episode-specific routes.
5. Only patch gaps in the target skill -- do not duplicate existing
content.

You will be told the maximum number of edits (the budget L). Produce
AT MOST L edits. You may produce fewer if warranted.
```

```
Every non-add edit MUST include `edit_skill` and it must name exactly
one existing skill. Also include `target_field` (default `body`) and
`operation_scope` (`skill`).

## Generalization Requirements

Do not memorize this minibatch. Convert segment evidence into reusable
ALFWorld decision rules.

A valid edit should:
- describe a decision condition, not a specific episode path
- address a reusable failure mode, not repair one exact segment
- avoid concrete object names, receptacle names, room layouts, and
instance numbers unless the rule applies to the entire ALFWorld task
family
- use generic categories such as target object, source receptacle,
destination receptacle, container, appliance, inventory, and observed
feedback
- include when to stop applying the rule or when to transition to the
next phase
- preserve existing broadly useful behavior

Prefer no edit over an unsupported, episode-specific, or ambiguous
edit.

Respond ONLY with a valid JSON object (no markdown fences, no extra
text):
{
  "batch_size": <number of segments analysed>,
  "contribution": "harmful",
  "failure_summary": [
    {"failure_type": "<type>", "count": <int>, "description":
    "<one-line>"}
  ],
  "patch": {
    "reasoning": "<why these edits address the batch's common local
    failures>",
    "edits": [
      {"op": "append", "edit_skill": "<existing skill id>",
      "target_field": "body", "operation_scope": "skill", "content":
      "<markdown to add at end of Strategy Steps, before
      Termination>"},
      {"op": "insert_after", "edit_skill": "<existing skill id>",
      "target_field": "body", "operation_scope": "skill", "target":
      "<exact heading/text to insert after>", "content":
      "<markdown>"},
      {"op": "replace", "edit_skill": "<existing skill id>",
      "target_field": "body", "operation_scope": "skill", "target":
      "<exact text to replace>", "content": "<replacement>"},
      {"op": "delete", "edit_skill": "<existing skill id>",
      "target_field": "body", "operation_scope": "skill", "target":
      "<exact text to remove>"}
    ]
  }
}
```

D.5 SEGMENT POLARITY JUDGE

The LLM-polarity baseline uses this template to judge the target segment. The judge is not given C_i, A_i, or S_i. Its decision is based on the focused steps; any surrounding steps serve as context.

Segment polarity judge

```
You are an expert analyst for multi-skill ALFWorld agents.

You will judge ONE skill segment inside a longer trajectory. Judge
whether the focused steps were helpful, harmful, mixed, or
insufficient.

## Hard rules
1. Judge only the target skill on the focused steps.
2. Non-focused steps are context only; do not treat them as sufficient
evidence to praise or blame the target skill.
3. Do NOT infer that the skill caused final task success or failure
merely because it was active.
4. Do NOT invent environment facts that are not shown in the provided
actions/feedback.
5. If evidence is weak, conflicting, or outside this skill's stated
responsibility, use `mixed` or `insufficient`.

## Rubric
- `helpful`: focused actions/feedback show the skill made useful
progress toward its own responsibility.
- `harmful`: focused actions/feedback show the skill caused avoidable
local mistakes, loops, or bad decisions in its responsibility.
- `mixed`: both helpful and harmful local evidence, or local progress
with material drawbacks.
- `insufficient`: too little focused evidence, or the outcome is
dominated by other skills / later phases.

Respond ONLY with a JSON object:
{
  "label": "helpful|harmful|mixed|insufficient",
  "score": <-2|-1|0|1|2>,
  "confidence": <0.0-1.0>,
  "evidence_steps": [<int>, ...],
  "rationale": "<2-4 sentences citing focused actions/feedback>"
}
```

D.6 Shared Ablation Editor

The episode-label, C_i-only, LLM-polarity, and SkillSeg configurations use the same editor prompt in the feedback ablations. Each configuration supplies its own segment polarity labels.

Shared ablation editor

```
You edit a frozen ALFWorld skill from signed segment feedback.
Positive evidence supports retaining useful instructions; negative
evidence supports revising instructions. Neutral evidence must not
justify edits. First write separate positive_reflection and
negative_reflection. Then propose at most 4 concise additions to the
skill body, grounded in the evidence and consistent with its
responsibility. Do not change termination, routing, or other skills.
Return JSON.
{"positive_reflection": "...", "negative_reflection": "...", "edits":
[{"instruction":"...", "evidence_segment_ids":["..."]}]}. Empty edits
are allowed. Avoid instructions specific to a single object ID or
room. Only use supplied segment IDs.
```

The prompt remains fixed across these configurations, allowing the ablations to compare the sources of feedback labels.

D.7 Candidate Screening Interface

Candidate screening uses likelihood replay without an additional generative prompt. It takes the current and candidate skill libraries, historical states, recorded actions, and their advantages. Both libraries are scored on the same states and actions. Appendix A gives the screening rule and the definition of J.

D.8 Ablation Feedback Inputs

The segment-label and likelihood-only variants use the shared editor in Appendix D.6 without separate generative prompts. The segment-label variant copies the episode label to each segment, while the likelihood-only variant uses $\mathrm{sign}(C_i)$. These variants change the feedback source while keeping the editor prompt fixed.

E Qualitative Analysis of Skill Updates

This appendix presents training cases that illustrate how SkillSeg assigns feedback to localized skill segments and produces targeted skill patches. The cases cover a positive segment in a failed trajectory, a negative segment in a successful trajectory, and different feedback assignments within one successful trajectory.

For each segment, the feedback is determined by $S_i = C_i A_i$, where C_i is the skill-conditioned likelihood contrast and A_i is the trajectory outcome relative to its historical baseline. Reflection groups segments by target skill and polarity before generating a patch. The patches shown below are produced from the corresponding positive or negative bucket for that skill and may combine evidence from multiple segments in the same evolution iteration.

E.1 Positive Feedback in a Failed AppWorld Trajectory

This case comes from the tenth evolution iteration on AppWorld. The task required changing one alarm and disabling the others:

> **Task:** I am going on a vacation. Move my wake-up phone alarm to 40 minutes earlier and disable the rest.

The task eventually failed, but a consecutive segment executed by the skill `react_feedback_decision` successfully completed two alarm-disabling operations.

Table 9: A positive segment inside a failed AppWorld trajectory.

Field	Value
Skill	`react_feedback_decision`
Step range	9–10
Number of actions	2
Episode outcome	Failure
C_i	-1.3505
A_i	-0.1818
S_i	$+\mathbf{0.2456}$
Polarity bucket	Positive

The recorded actions and environment feedback are shown below. Ellipses abbreviate omitted call arguments.

Alarm updates and environment feedback

```
Action 1:
print(apis.phone.update_alarm(alarm_id=417, ... , enabled=False))

Environment feedback:
```

```
{"message": "Alarm updated successfully."}

Action 2:
print(apis.phone.update_alarm(alarm_id=419, ... , enabled=False))

Environment feedback:
{"message": "Alarm updated successfully."}
```

Both alarm updates returned a successful environment message. Here, $C_i < 0$ and $A_i < 0$, giving a positive S_i and routing the segment to the positive feedback bucket for its target skill.

The positive bucket for the skill `react_feedback_decision` produced the following patch:

Positive patch for react_feedback_decision

```
- When a task requires information from multiple apps or data sources,
  retrieve contextual data from all relevant sources before executing
  task-specific actions.

- After collecting data from multiple sources, verify that all
required
  sources have been covered before terminating.
```

The patch summarizes information-collection and completeness checks from the positive feedback bucket. The candidate received gate score $J = 0.2560$ and was accepted. This case shows that a failed episode does not force every skill segment into the negative feedback bucket.

E.2 Negative Feedback in a Successful ALFWorld Trajectory

This case comes from the thirteenth evolution iteration on ALFWorld. All five rollouts in the iteration succeeded. The task was:

> **Task:** put a cool apple in countertop.

Although the task succeeded, an initial exploration segment executed by the skill `explore_exploit_arbitration` was assigned to the negative feedback bucket.

Table 10: A negative segment inside a successful ALFWorld trajectory.

Field	Value
Skill	`explore_exploit_arbitration`
Step range	0–1
Number of actions	2
Episode outcome	Success
C_i	-0.1692
A_i	$+0.2727$
S_i	$\mathbf{-0.0462}$
Polarity bucket	Negative

The recorded actions and environment feedback are:

Exploration actions and environment feedback

```
Action 1:
go to cabinet 1

Environment feedback:
You arrive at cabinet 1. On the cabinet 1, you see a bowl 1.

Action 2:
```

```
go to fridge 1

Environment feedback:
You arrive at fridge 1. The fridge 1 is closed.
```

The segment has $C_i < 0$ and $A_i > 0$, so $S_i < 0$. SkillSeg therefore sends it to negative reflection instead of copying the successful episode label to the segment.

The negative bucket for the skill `explore_exploit_arbitration` produced the following patch:

Negative patch for explore_exploit_arbitration

```
- Complete transformation before taking: When the task requires
  transforming an object, do not take the object from a transformation
  appliance until the transformation action has been performed.

- If an object has been placed in an appliance, perform the required
  transformation, such as heating or cooling, before taking it out.

- Verify the object state before moving to the placement phase.
```

The patch adds ordering checks for object transformation and placement. The candidate received gate score $J = 0.3487$ and was accepted. This case shows that a successful episode does not make every participating skill segment positive by default.

E.3 Different Feedback Assignments within One Successful ALFWorld Trajectory

This case comes from the third evolution iteration on ALFWorld. All five rollouts in the iteration succeeded, and the candidate gate score was $J = 0.3663$. One successful trajectory had the following task:

Task: put some alarmclock on desk.

The four localized segments below share the same successful episode outcome but receive different feedback assignments.

Table 11: Different feedback assignments within one successful ALFWorld trajectory.

Segment skill	Step range	C_i	A_i	S_i	Polarity
`strategic_planning`	0–1	+0.5007	+0.4286	+0.2146	Positive
`react_feedback_decision`	2–3	+0.8117	+0.4286	+0.3479	Positive
`structured_reasoning`	4–4	-0.1313	+0.3333	-0.0438	Negative
`strategic_planning`	7–8	+0.1982	+0.4286	+0.0849	Positive

The positive bucket for the skill `strategic_planning` produced the following patch:

Positive patch for strategic_planning

```
- Maintain a record of already-searched locations to avoid redundant
searches.

- When the target object is not found in initial locations,
systematically
  check alternative locations based on object type context.

- Understand the complete task flow:
  find -> transform or cool -> place.
```

The negative bucket for the skill `structured_reasoning` produced the following patch:

Negative patch for structured_reasoning

```
Pre-execution verification:

Before outputting the selected action, verify that it appears in the
admissible actions list. If the action is not admissible, restart the
reasoning process with updated constraints.
```

The two patches target different skills and encode different update content. The first reinforces search records and task-phase planning, while the second adds an admissibility check before action generation. Thus, segments in one successful trajectory can provide different editing directions even when their episode label is identical.

E.4 SUMMARY

These cases illustrate the path from segment scoring to skill-specific patch generation. A failed episode can contain a segment assigned to positive feedback, a successful episode can contain a segment assigned to negative feedback, and different segments in one successful trajectory can receive different polarities. SkillSeg therefore uses the episode outcome as context while routing localized behavioral evidence to the corresponding skill update.