From Failure to Foresight: Temporal Anticipatory Reflective Optimization for Skill Learning in Agents

Anonymous ACL submission

Abstract

Large language model agents use procedural skills to solve long-horizon tasks, but corrections discovered after execution failures may depend on feedback unavailable when a skill is first generated. We propose **Temporal Anticipatory Reflective Optimization (TARO)**, a reinforcement learning framework that trains a skill generator while keeping the solver frozen. During training, execution-grounded refinement explores alternative skills following an initial failure. **Retrospective flattening** then pairs these executed variants with the initial failure context and learns from their relative rewards through a clipped counterfactual-distillation objective. The update uses a fixed pre-update likelihood under the training context as its anchor; it is not an importance-sampling correction for the contexts that generated the refinements. At evaluation, the generator produces one skill for a single retry after an unguided failure. Experiments on WebShop and ALFWorld show improvements over the evaluated controls in aggregate success or score, while comparisons with published skill-learning results require attention to their different protocols.

1 Introduction

Large language model (LLM) agents (Wang et al., 2024a) have recently demonstrated strong capabilities in complex tasks such as web navigation and tool use (Yao et al., 2022b), largely by decomposing high-level goals into executable action sequences. To improve generalization and support reuse across tasks, recent agent systems have increasingly incorporated *skills* (Anthropic, 2024), i.e., reusable procedural abstractions that encapsulate task-solving strategies and can be retrieved, invoked, or adapted in new environments.

Early skill-based agents typically relied on static or manually curated skill repositories. In such systems, skills are predefined and primarily reused through retrieval or selection, which limits the agent's ability to acquire new procedural knowledge from interaction experience. To address this limitation, recent work has explored dynamic skill evolution (Xia et al., 2026; Alzubi et al., 2026), where agents iteratively expand, revise, or reorganize their skill libraries based on task feedback, execution trajectories, and failure cases. Although these methods improve the adaptability of skill libraries, they mainly operate at the level of library maintenance rather than directly training a model to generate and optimize reusable skills. This has motivated a more recent line of work that explicitly models the skill-generation process itself, leading to dedicated training frameworks for a *skill generator* that can autonomously compose, revise, and curate skills (Wu et al., 2025a; Ouyang et al., 2026).

Learning from downstream outcomes is already central to skill curation: SkillOS trains a curator

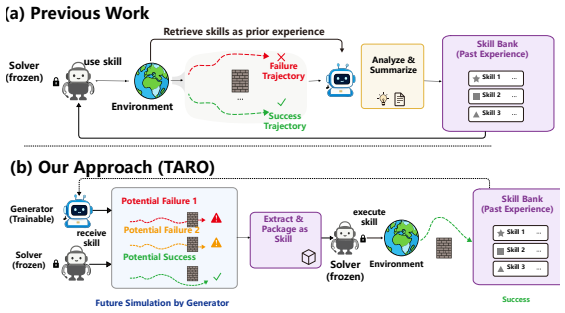


Figure 1: **Retrospective skill reuse and TARO. (a)** A schematic of trajectory-based skill construction: summaries of completed executions are stored and retrieved for later tasks. **(b)** After an initial failure, the generator proposes and revises skills using feedback from frozen-solver environment rollouts. This training-time execution is labeled "future simulation": the generator supplies skills, while the solver performs the rollouts. Retrospective flattening trains these variants under the initial failure context for one skill-guided retry at evaluation.

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.

to improve a repository using performance on subsequent tasks (Ouyang et al., 2026). We study a complementary question within a failed task: how can a generator internalize corrections discovered through several execution–feedback rounds, so that it can later produce useful guidance from the initial failure context alone? Training revised skills only under their richer refinement prompts leaves this deployment context mismatch unresolved.

Motivated by this question, we propose **Temporal Anticipatory Reflective Optimization (TARO)** (Figure 1). Training-time solver executions expose weaknesses in candidate skills, and retrospective flattening transfers reward-ranked revisions back to the initial failure context. Here, anticipation refers to applying learned corrections before the skill-guided retry; it does not imply an explicit world model or prevention before any failure has been observed. Our primary contributions are summarized as follows:

- We propose **TARO**, a framework that trains procedural skill generation through execution-grounded refinement while retaining a frozen solver and a single skill-guided retry at evaluation.
- We formulate **retrospective flattening** as counterfactual distillation with signed group-relative advantages and PPO-style clipping. The objective explicitly separates the sampling context from the training-context likelihood anchor.
- We evaluate aggregate performance, transfer, and skill quality on WebShop and ALFWorld, and compare with published SkillOS results in the main table, identifying their original source.

2 Preliminaries

We first formalize the dual-agent skill-learning setting and then revisit the retrospective skill-construction paradigm adopted by existing skill-based agents.

2.1 Problem Formulation

Let $x \in \mathcal{D}$ denote a task instruction sampled from an environment distribution. We consider a frozen solver policy $\pi_s : \mathcal{H} \to \mathcal{A}$, which maps an interaction history $\mathcal{H}$ to an atomic environment action $a \in \mathcal{A}$, such as `search[]` or `click[]` in WebShop (Yao et al., 2022a). Throughout training, the solver remains fixed, and only the skill-generation policy is optimized.

We introduce a trainable skill generator $\pi_\theta(z \mid C)$, which generates a structured procedural skill $z \in \mathcal{Z}$ conditioned on a textual context C. The generated skill is prepended to the solver prompt and serves as high-level procedural guidance for subsequent environment interaction. Given a task x and a skill z, the frozen solver interacts with the environment and produces a trajectory:

$$\begin{aligned}\tau &= (o_1, a_1, r_1, \ldots, o_T, a_T, r_T) \\ &\sim p(\tau \mid x, z; \pi_s),\end{aligned} \tag{1}$$

where o_t, a_t, and r_t denote the observation, action, and step-level reward at time step t. The environment provides a sparse terminal reward $R(x, z) \in \{0, 1\}$, where $R(x, z) = 1$ indicates task success. We use $z = \varnothing$ to denote the unguided setting in which no external skill is provided to the solver.

The objective is to optimize the skill generator such that its generated skills improve the task success rate of the frozen solver:

$$\max_\theta \ \mathbb{E}_{x \sim \mathcal{D},\, z \sim \pi_\theta(\cdot|C)} \left[R(x, z) \right]. \tag{2}$$

This formulation differs from directly training an action-level agent: the trainable policy does not execute environment actions itself, but instead learns to generate reusable procedural abstractions that modulate the behavior of a fixed downstream solver.

2.2 Retrospective Skill Construction

Existing skill-based agents typically construct skills in a retrospective manner. Given a failed trajectory τ^-, they summarize the observed failure into a reusable skill:

$$z = \mathcal{S}(x, \tau^-, \mathcal{B}), \tag{3}$$

where $\mathcal{S}$ denotes a skill summarization function and $\mathcal{B}$ is the current skill bank. The resulting skill is stored and later retrieved for semantically similar tasks.

This formulation describes heuristic retrospective construction, rather than every skill-learning method. Learned curators such as SkillOS additionally optimize repository operations through downstream rewards (Ouyang et al., 2026). TARO instead concentrates on training a generator to reproduce useful within-task refinements from the initial failure context.

3 Methodology

3.1 Overview of TARO

TARO (Figure 2) trains a skill generator using rewards from a frozen solver. It collects skills under progressively refined prompts and then optimizes those skills under the initial failure prompt. We distinguish the generator's actual decoding distribution q_b from the likelihoods used during the actor update; q_b includes the sampling settings in Appendix A.1.

TARO consists of three stages. First, in *failure-conditioned skill generation*, the frozen solver is executed without skill guidance, and only tasks that the solver fails are selected for skill generator training. This focuses optimization on cases where additional procedural guidance is actually needed. Second, in *execution-grounded temporal refinement*, the skill generator generates candidate skills, observes their execution outcomes through the frozen solver, and iteratively revises unsuccessful skills according to trajectory-level feedback. This stage produces executed skill variants with success or failure feedback. Third, in *retrospective flattening and policy optimization*, all executed skill variants are mapped back to the initial failure context and optimized with a signed group-relative surrogate. The training objective encourages useful revisions under the context available during the standard single-retry protocol.

3.2 Failure-Conditioned Skill Generation

TARO trains the skill generator only on tasks where the frozen solver fails without skill guidance. For each task x, we first perform an unguided rollout:

$$\tau_0 \sim p(\tau \mid x, \varnothing; \pi_s). \tag{4}$$

If the solver succeeds, i.e., $R(x, \varnothing) = 1$, the task is excluded from skill generator optimization because no additional procedural intervention is required. This filtering strategy avoids training the skill generator on redundant tasks and concentrates learning on failure cases where skill generation can provide meaningful improvement.

For each failed task, a failure summarizer F compresses the trajectory into diagnostic feedback:

$$d_0 = F(x, \tau_0) = (\hat{\tau}_0, e_0), \tag{5}$$

where $\hat{\tau}_0$ is an abstracted action trace and e_0 is a behavioral diagnosis describing the likely cause of failure. We then construct the initial failure context:

$$C_0 = \mathrm{Prompt}_{\mathrm{gen}}(x, \hat{\tau}_0, e_0, \mathcal{B}_k(x)), \tag{6}$$

where $\mathcal{B}_k(x)$ denotes the top-k retrieved skills from the skill bank $\mathcal{B}$. The skill bank stores previously successful skills and retrieves relevant ones according to semantic similarity.

Conditioned on C_0, the skill generator samples a group of candidate skills:

$$z_i^{(1)} \sim q_b(\cdot \mid C_0), \quad i = 1, \ldots, G. \tag{7}$$

Algorithm 1 Temporal Anticipatory Reflective Optimization (TARO)

Require: Task dataset $\mathcal{D}$, frozen solver π_s, skill generator π_θ, skill bank $\mathcal{B}$, group size G, refinement depth K
Ensure: Optimized skill generator π_θ

for each task $x \in \mathcal{D}$ **do**
  Roll out the solver without skill guidance and obtain τ_0
  if $R(x, \varnothing) = 1$ **then**
    continue
  end if
  Construct initial failure context C_0 from $(x, \tau_0, \mathcal{B})$
  Sample initial skills $\{z_i^{(1)}\}_{i=1}^{G} \sim q_b(\cdot \mid C_0)$
  Initialize variant set $\mathcal{V}_x \leftarrow \emptyset$
  for $i = 1, \ldots, G$ **do**
    for $k = 1, \ldots, K$ **do**
      Execute $z_i^{(k)}$ with π_s and obtain reward $r_i^{(k)}$
      Add $(z_i^{(k)}, r_i^{(k)})$ to $\mathcal{V}_x$
      if $r_i^{(k)} = 1$ **then**
        Add $z_i^{(k)}$ to $\mathcal{B}$ and **break**
      end if
      if $k < K$ **then**
        Generate refined skill $z_i^{(k+1)}$ using execution feedback
      end if
    end for
  end for
  Flatten $\mathcal{V}_x$ into counterfactual samples under C_0
  Recompute fixed π_{old} token likelihoods under C_0
  Update with signed advantages and Equation 18
end for

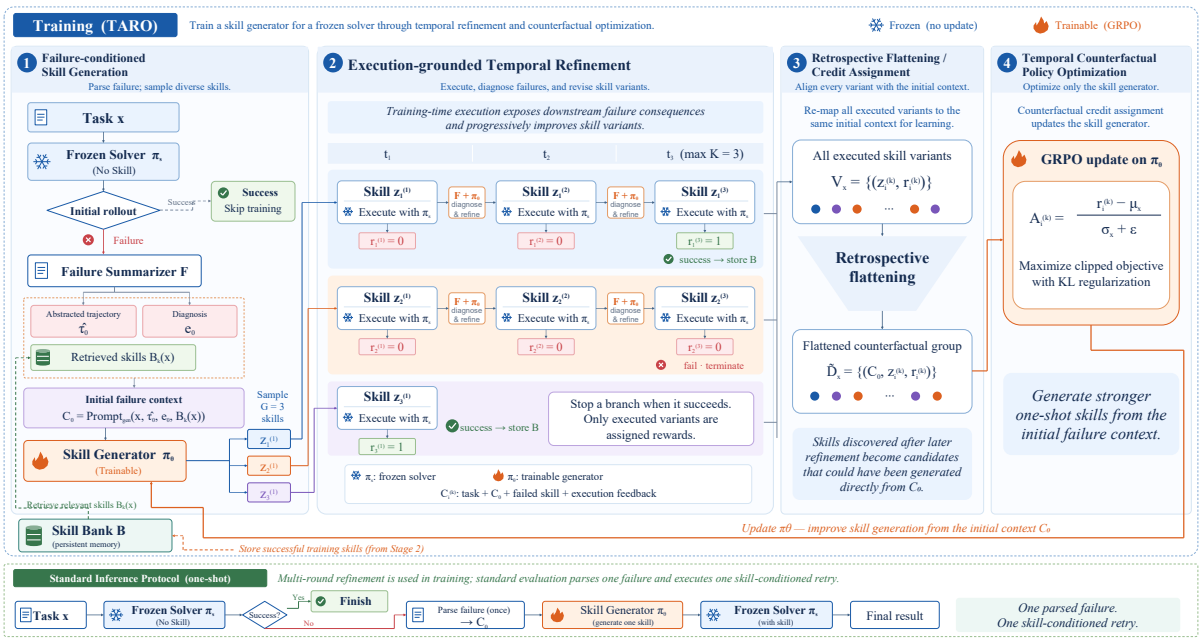


Figure 2: **Overview of TARO.** An unguided failure is parsed into the initial context C_0, from which the generator samples G skill branches. The frozen solver executes each candidate; unsuccessful skills are revised using execution feedback for up to K rounds, with early stopping on success. Retrospective flattening pairs all executed skill–reward variants with C_0 for training while retaining their observed rewards. This changes the training context, not the sampling process. Signed group-relative advantages and the clipped counterfactual-distillation objective (Equation 18) update only the generator. Standard evaluation uses one parsed failure and one skill-guided retry.

Sampling multiple skills under the same failure context allows TARO to compare alternative procedural strategies for the same task. This group structure later provides the basis for relative reward normalization and stable policy optimization.

3.3 Execution-Grounded Temporal Refinement

Each generated skill is immediately evaluated through the frozen solver. Given a candidate skill $z_i^{(k)}$, the solver produces a trajectory:

$$\tau_i^{(k)} \sim p(\tau \mid x, z_i^{(k)}; \pi_s), \tag{8}$$

and receives a terminal reward:

$$r_i^{(k)} = R(x, z_i^{(k)}). \tag{9}$$

If $r_i^{(k)} = 1$, the skill is considered successful and is added to the skill bank for future retrieval. If $r_i^{(k)} = 0$, TARO performs temporal refinement by using the failed execution to construct a more informative refinement context.

Specifically, the failed skill-conditioned trajectory is summarized as:

$$d_i^{(k)} = F(x, z_i^{(k)}, \tau_i^{(k)}) = (\hat{\tau}_i^{(k)}, e_i^{(k)}), \tag{10}$$

where $\hat{\tau}_i^{(k)}$ abstracts the solver's behavior under the current skill and $e_i^{(k)}$ diagnoses the remaining failure. A refinement context is then constructed as:

$$C_i^{(k)} = \mathrm{Prompt}_{\mathrm{ref}}(x, C_0, z_i^{(k)}, \hat{\tau}_i^{(k)}, e_i^{(k)}). \tag{11}$$

The skill generator generates a revised skill:

$$z_i^{(k+1)} \sim q_b(\cdot \mid C_i^{(k)}). \tag{12}$$

This execution–reflection–revision loop continues until the skill succeeds or the maximum refinement depth K is reached.

Temporal refinement is used during training to collect skill variants that encode different degrees of downstream failure awareness. TARO does not perform multi-round test-time refinement in the standard setting. It uses one parsed failed rollout to generate a single skill-conditioned retry. Multi-round test-time refinement is evaluated only as an explicitly labeled comparison variant.

3.4 Retrospective Flattening

Set $C_i^{(0)} = C_0$. The actual collection process samples

$$z_i^{(k)} \sim q_b(\cdot \mid C_i^{(k-1)}), \tag{13}$$

where q_b includes the decoding configuration. Later skills therefore depend on feedback available only after previous executions. Let $\mathcal{V}_x$ contain all executed pairs $(z_i^{(k)}, r_i^{(k)})$, with $i = 1, \ldots, G$ and $k = 1, \ldots, K_i$. The number of executed variants is $M_x = \sum_i K_i$, with $K_i \leq K$ determined by stopping. Retrospective flattening constructs

$$\widetilde{\mathcal{D}}_x = \{(C_0, z, r) \mid (z, r) \in \mathcal{V}_x\}. \tag{14}$$

This changes the context used for training, not the distribution that generated the skill. Rewards remain the observed outcomes of executing each skill with the frozen solver. Counterfactual training means learning to propose the same skill under C_0, without claiming that it was originally sampled there.

The motivation is to transfer useful refinements into the generator used for a single retry. Refinement can introduce reward contrast when a later skill succeeds after initial candidates fail. It does not guarantee nonzero advantages: if all executed variants receive the same reward, the centered reward signal remains zero.

3.5 Clipped Counterfactual Distillation

We retain group-relative reward normalization and PPO-style clipping from GRPO (Shao et al., 2024; Schulman et al., 2017), while changing their interpretation for the flattened collection. For every executed variant,

$$A_i^{(k)} = \frac{r_i^{(k)} - \mu_x}{\sigma_x + \epsilon_0}, \tag{15}$$

where μ_x and σ_x are computed over $\mathcal{V}_x$. Let π_{old} be the fixed generator snapshot evaluated immediately before the actor update. For response token $y_{i,t}^{(k)}$ of skill $z_i^{(k)}$, define the anchored ratio

$$\rho_{i,t}^{(k)}(\theta) = \frac{\pi_\theta(y_{i,t}^{(k)} \mid C_0, y_{i,<t}^{(k)})}{\pi_{\text{old}}(y_{i,t}^{(k)} \mid C_0, y_{i,<t}^{(k)})}. \tag{16}$$

Both likelihoods use C_0; the denominator is an update anchor, not the refined skill's likelihood under its actual sampling context. In particular, it does not correct $q_b(\cdot \mid C_i^{(k-1)})$ into $\pi_\theta(\cdot \mid C_0)$.

The token-level surrogate is

$$\ell_{i,t}^{(k)}(\theta) = \min\big[\rho_{i,t}^{(k)}(\theta) A_i^{(k)}, \; \text{clip}(\rho_{i,t}^{(k)}(\theta), 1-\epsilon, 1+\epsilon) A_i^{(k)}\big]. \tag{17}$$

Advantages and anchor likelihoods remain fixed within the update. The empirical objective is

$$\widehat{\mathcal{J}}_{\text{TARO}}(\theta) = \text{Agg}_{\text{mask}}\left[\ell_{i,t}^{(k)}(\theta) - \beta_{\text{KL}} d_{i,t}^{(k)}(\theta)\right], \tag{18}$$

where Agg_{mask} denotes the trainer's aggregation over valid response tokens in the collected batch. The nonnegative reference-policy penalty d is defined in Appendix A.1; its evaluation on refined tokens is an empirical regularizer, rather than an asserted unbiased estimate of sequence-level KL.

We interpret the update as *counterfactual distillation with signed group-relative advantages*. Positive advantages promote useful refinements under C_0, and negative advantages suppress unsuccessful variants. This retains a reward-driven training signal while avoiding a claim of unbiased off-policy policy gradients. Merely replacing the denominator with a refinement-context likelihood would not make the existing clipped token objective unbiased; that would require a separate derivation accounting for the actual decoding distribution, adaptive collection, and aggregation.

4 Experiments

We evaluate performance across frozen solvers, transfer to held-out task structures, component attribution, refinement internalization, training cost, and generated-skill quality.

4.1 Experimental Setup

WebShop (Yao et al., 2022a) uses the standard split of 10,587 training, 1,000 development, and 500 test tasks and reports environment score and success rate (SR). ALFWorld (Shridhar et al., 2021) reports SR over 140 tasks. The generator is initialized from Qwen3-8B-Instruct; Qwen3-8B, Qwen3-32B (Yang et al., 2025), and Gemini-2.5-Pro (Comanici et al., 2025) are frozen solvers. Standard evaluation retains an unguided success; otherwise, one parsed failed rollout conditions a single skill-guided retry, with no multi-round test-time refinement. Training uses $G = 3$ branches, at most $K = 3$ executed skills per branch, and the GRPO-inspired surrogate with learning rate 1×10^{-6}, batch size 128, and 16 gradient-accumulation steps. **Base Generator** is the frozen Qwen3-8B-Instruct initialization. **Reflection** adds iterative execution–feedback–revision at evaluation; **Skill Memory** adds retrieval from the frozen seen-task bank. Unavailable full-set measurements for these two comparisons are not shown in

the main table. Prompts, decoding, parser boundaries, and reproducibility details remain in Appendix A.

4.2 Main Results

Table 1 reports the complete task-type breakdown. Among locally evaluated methods, TARO achieves the highest ALFWorld average SR and WebShop score with every solver. With Qwen3-32B, it exceeds GRPO by 10.7 ALFWorld points and 6.6 WebShop-score points. Gains span multiple ALFWorld categories rather than a single task type, although TARO does not lead every category. Gemini WebShop SR remains slightly below Base Generator (41.4 vs. 41.6).

The GRPO baseline uses a training skill-rollout budget aligned with TARO (Appendix A.9).

Comparison with SkillOS. Table 1 includes SkillOS results quoted from the original paper (Ouyang et al., 2026). For Qwen3-8B, Qwen3-32B, and Gemini-2.5-Pro, respectively, TARO's ALFWorld overall SR is 1.6, 1.2, and 0.5 percentage points below SkillOS. On WebShop, TARO's scores are 0.2, 0.3, and 0.2 points higher, while its SR differences are -0.1, -0.1, and $+0.1$ percentage points. The small score gains therefore do not correspond to a consistent increase in task completion. SkillOS retains higher reported overall success on ALFWorld, while WebShop success rates are numerically close. These are comparisons of reported values under different evaluation protocols, detailed in Appendix A.8.

4.3 Held-Out Generalization

WebShop generalization is evaluated by five-fold leave-one-category-out over the goal records' coarse `category` field: `beauty`, `electronics`, `fashion`, `garden`, and `grocery`. Each category is held out once, with the other four serving as seen categories. ALFWorld holds out task–object–receptacle/room combinations. Splits are fixed before training; the evaluation bank is frozen to seen-task skills, and held-out skills are never written back. Full split and leakage controls are in Appendix A.2.

TARO obtains the strongest held-out results and smallest degradation (Table 2): drops are 4.4/3.6 points, versus 7.2/5.5 for Base Generator and 6.1/4.9 for Reflection. Thus, evaluation-time revision alone does not account for the transfer gains.

4.4 Component Attribution

Full TARO combines trajectory-specific feedback, temporal refinement, RF, and the clipped surrogate. Without RF, refined skills are optimized under their chronological contexts; without refinement, only initial candidates are used. Generic feedback replaces the diagnosis with a fixed failure message; final-reward-only training removes both feedback and refinement. Reusing exactly the same refinement data and parser outputs, RF raises Qwen3-8B score/SR from 33.8/10.4 to 40.8/16.4 and Qwen3-32B from 40.4/13.0 to 49.5/16.4 (Table 3). The primary gain therefore cannot be attributed to a changed external parser.

4.5 Refinement Internalization

Standard TARO slightly exceeds iterative Base Generator on ALFWorld SR (59.8 vs. 59.3) and WebShop score (40.8 vs. 40.3), using one skill rollout and one parser call rather than up to three of each (Table 4). Additional iterative evaluation still improves TARO, but training internalizes most of the observed refinement benefit. Cost accounting and evaluation-set distinctions are detailed in Appendix A.5.

4.6 Refinement Depth and Training Efficiency

Both solvers improve through $K = 3$. Moving to $K = 4$ adds only 0.4/0.6 WebShop SR points, while time per parameter update rises from 1492/1613 to 2135/2311 seconds (Table 5); we therefore use $K = 3$.

Adaptive stopping additionally rejects a near-duplicate revision before execution when normalized token-level edit distance is below $\delta = 0.05$, without an extra model or API call. Unexecuted candidates enter neither GRPO nor the bank. It reduces average executed skill rollouts per branch by 15.0% and time per parameter update from 1492 to 1305 seconds, with unchanged ALFWorld SR and a 0.2-point decrease in both WebShop metrics (Table 6). The distance definition and matched settings are in Appendix A.3.

4.7 Generated-Skill Quality

Two annotators assess paired skills from Base Generator and TARO on 100 held-out tasks (50 per benchmark), with disagreements adjudicated. Table 7 reports all benchmark-level rates; raw counts and the binary rubric remain in Appendix A.6.

Table 1: Results with three frozen solvers. Bold marks the largest reported value per solver. †: SkillOS results quoted from Ouyang et al. (2026) under its original protocol (Appendix A.8); other rows are evaluated in this work.

Frozen solver	Method	ALFWorld SR (%)							WebShop	
		Pick	Look	Clean	Heat	Cool	Pick2	Avg.	Score	SR (%)
Qwen3-8B	No Skill	78.3	45.9	33.5	37.8	29.5	47.0	47.9	32.1	10.0
	Base Generator	82.4	54.1	46.5	41.0	40.2	56.3	56.1	39.4	14.2
	GRPO	78.5	43.1	38.0	36.2	34.5	50.2	49.9	34.2	10.2
	SkillOS†	**85.7**	**56.4**	**54.3**	**43.8**	**46.7**	**62.5**	**61.2**	40.6	**16.5**
	TARO	85.6	51.2	53.8	39.5	45.2	61.1	59.6	**40.8**	16.4
Qwen3-32B	No Skill	80.6	69.8	45.9	37.7	42.9	43.4	54.9	41.9	12.4
	Base Generator	**94.5**	71.3	51.2	54.2	46.8	55.0	64.1	46.8	14.6
	GRPO	81.2	66.4	47.2	43.0	44.1	48.5	56.7	42.9	12.0
	SkillOS†	91.4	**76.9**	**59.3**	**56.3**	**57.3**	**62.5**	**68.6**	49.2	**16.5**
	TARO	90.8	76.2	58.4	51.0	55.3	62.1	67.4	**49.5**	16.4
Gemini-2.5-Pro	No Skill	90.5	66.7	48.1	39.6	68.0	68.1	66.4	48.6	38.4
	Base Generator	92.5	**74.1**	70.3	**73.5**	75.2	69.8	77.4	54.9	**41.6**
	GRPO	91.0	67.2	54.3	48.5	71.2	65.4	68.9	52.3	39.2
	SkillOS†	**95.2**	71.8	74.1	72.9	77.3	**77.8**	**80.2**	56.0	41.3
	TARO	94.8	71.0	**74.2**	70.1	**78.3**	76.5	79.7	**56.2**	41.4

Table 2: Seen-to-held-out transfer. Drop is the absolute decrease in ALFWorld SR (percentage points) or WebShop score (points); lower is better.

Method	Seen	Held-out	Drop
	ALFWorld SR (%)		
Base Generator	54.8	47.6	7.2
Reflection	59.3	53.2	6.1
TARO	**59.8**	**55.4**	**4.4**
	WebShop Score		
Base Generator	36.2	30.7	5.5
Reflection	40.3	35.4	4.9
TARO	**40.8**	**37.2**	**3.6**

Table 3: WebShop attribution with both Qwen solvers. RF denotes retrospective flattening. All rows use group-relative advantages and clipping; full control definitions are in Appendix A.4.

Variant	Qwen3-8B		Qwen3-32B	
	Score	SR (%)	Score	SR (%)
Full TARO	**40.8**	**16.4**	**49.5**	**16.4**
Without RF	33.8	10.4	40.4	13.0
Without refinement	31.3	10.6	39.7	11.2
Generic feedback	37.5	13.8	46.2	14.6
Final reward only	34.2	10.2	42.9	12.0

TARO's largest overall gains are in specificity (+20 points), usefulness (+18), and non-hallucination (+16). Usefulness improves by 18 points on each benchmark, while executability remains high for both generators.

4.8 Skill-Bank Growth and a Refinement Case

The bank grows from 0 to 68 ALFWorld skills and 56 WebShop skills by training step 150 (Figure 3). This documents accumulation but does not establish that a larger bank causes higher SR. Table 8 illustrates how a missed constraint becomes an explicit verification step in a refined skill.

5 Related Work

Self-Reflection and Skill-Based Memory in Agents. While large language models enable increasingly complex sequential decision-making (Google, 2025; OpenAI, 2025) through paradigms like ReAct (Yao et al., 2022b), handling execution errors typically relies on ephemeral in-context self-reflection (Shinn et al., 2023; Wu et al., 2023; Dong et al., 2024). Such reflective reasoning often improves immediate task performance, but the resulting knowledge is rarely retained beyond the current interaction. To prevent agents from repeatedly encountering similar failures, recent literature augments them with

Table 4: Internalization on the split-controlled evaluation set, distinct from the full-set Base Generator row in Table 1. All variants start with an unguided attempt. Skill-rollout and parser-call budgets apply only after its failure; iterative evaluation stops on success.

Generator	Evaluation	Skill rollouts	Parser calls	ALFWorld Avg. SR (%)	WebShop Score	WebShop SR (%)
Base Generator	Standard	1	1	54.8	36.2	12.4
Base Generator	Iterative	1–3	1–3	59.3	40.3	15.9
TARO	**Standard**	**1**	**1**	**59.8**	**40.8**	**16.4**
TARO	Iterative	1–3	1–3	60.9	42.0	17.3

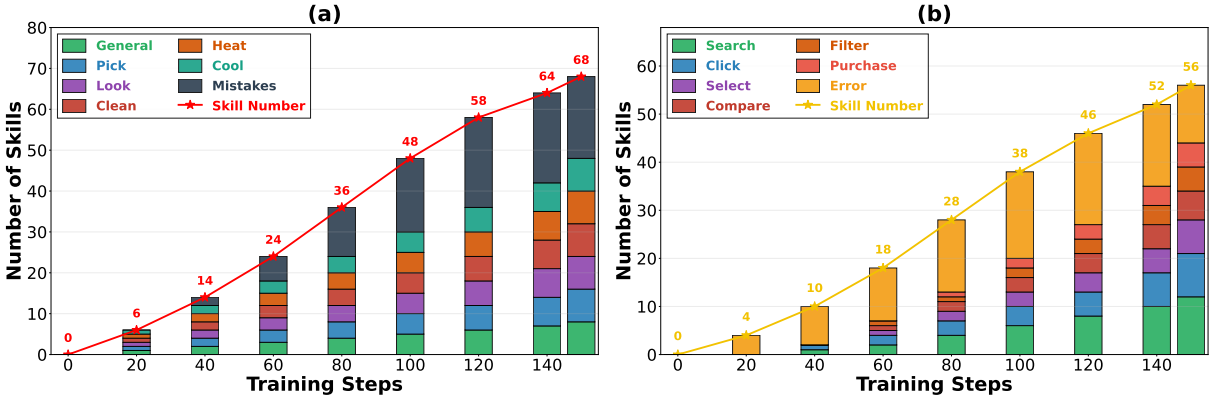


Figure 3: **Skill-bank growth during TARO training.** **(a)** ALFWorld; **(b)** WebShop. The axes show training steps and the number of stored skills. Stacked bars show the bank's composition by the displayed skill categories; lines and numeric labels indicate total bank size, reaching 68 and 56 at step 150, respectively. The plot describes accumulation; it does not isolate the effect of bank size on task success.

Table 5: Refinement depth on WebShop. K counts executed skills per training branch; $K = 1$ is one-shot, not no-skill. Time is measured in seconds per parameter update. Eff. is SR-point gain per 100 additional seconds per update over the preceding row.

K	Qwen3-8B			Qwen3-32B		
	SR (%)	Update time (s)	Eff.	SR (%)	Update time (s)	Eff.
1	10.6	483	–	11.2	524	–
2	13.8	1045	0.57	14.0	1139	0.46
3	16.4	1492	0.58	16.4	1613	0.51
4	16.8	2135	0.06	17.0	2311	0.09

reusable procedural memory (Wu et al., 2025b; Zhang et al., 2024). Moving beyond raw trajectories (Zheng et al., 2024) and workflows (Wang et al., 2024b), *skills* (Wang et al., 2025; Yang et al., 2026) have emerged as a foundational, agent-native memory format that supports compact storage and reusable behavioral abstraction. Rooted in embodied AI (Wang et al., 2023), these skills are now increasingly formalized into portable representations comprising instructions, metadata, and scripts (Anthropic, 2025), enabling stronger transferability across tasks and environments.

Table 6: Adaptive versus success-only stopping at $K_{\max} = 3$. Performance is measured under final one-shot evaluation. Time reports seconds per parameter update. The success-only column uses the standard TARO result. Rollout counts are averaged over training branches, and their reduction is relative to success-only stopping.

Metric	Success-only	Adaptive
ALFWorld SR (%)	59.6	59.6
WebShop Score	40.8	40.6
WebShop SR (%)	16.4	16.2
Avg. executed skill rollouts per branch	2.34	1.99
Rollout reduction	0%	15.0%
Time per parameter update (s)	1492	1305

Table 7: Positive skill-quality rates (%). Each cell is Base Generator / TARO; sample sizes are 50 WebShop tasks, 50 ALFWorld tasks, and 100 overall.

Criterion	WebShop	ALFWorld	Overall
Task relevance	78 / 88	80 / 90	79 / 89
Executability	86 / 90	88 / 92	87 / 91
Specificity	56 / 78	62 / 80	59 / 79
Non-hallucination	68 / 86	74 / 88	71 / 87
Usefulness	60 / 78	64 / 82	62 / 80

Table 8: Condensed WebShop refinement case, paraphrased from the recorded example rather than quoted verbatim.

Stage	Evidence
Task	Waterproof hiking backpack, price below $50, rating above 4.5.
Initial failure	Price checked but rating omitted; a product rated 3.8 was selected.
Parsed correction	Preserve all constraints; verify rating on the product page before purchase.
Refined skill	Check product type, waterproofing, price below $50, and rating above 4.5 before checkout.
Outcome	The retry selected a product rated 4.7 and completed the task.

Skill Curation: From Heuristics to Learning from Outcomes. Heuristic skill construction uses natural-language feedback and evolutionary search (Madaan et al., 2023; Alzubi et al., 2026). Learned approaches optimize skills or memory through task outcomes (Xia et al., 2026; Tu et al., 2026; Zhang et al., 2026). SkillOS explicitly trains a curator to insert, update, and delete repository skills using downstream task rewards while freezing the executor (Ouyang et al., 2026). TARO shares this separation of learning and execution, but studies distilling within-task refinement into generation from an initial failure context. Its distinction is the training-context reassignment, rather than the first use of downstream feedback for skill learning.

6 Conclusion

We introduced TARO for learning procedural guidance from execution-grounded refinement with a frozen solver. Retrospective flattening trains revised skills under the initial failure context using a clipped, signed-advantage surrogate. This formulation makes the context mismatch explicit and interprets the update as counterfactual distillation. Our local experiments show aggregate improvements over the evaluated baselines; the main table also includes published SkillOS results under its original protocol. The GRPO comparison aligns training skill-rollout budgets, while the fixed-data RF ablation isolates training-context assignment.

7 Limitations

While TARO demonstrates strong performance in enabling anticipatory skill synthesis for frozen solvers, it has several limitations that suggest directions for future work.

First, TARO still requires up to three sequential refinement executions per training branch. The evaluated adaptive-stopping rule reduces average executed skill rollouts per branch by 15.0%, but some unsuccessful branches may still revisit similar failure modes before reaching the stopping threshold.

Second, TARO requires sequential interaction with a frozen solver during training, as skill evaluation depends on executing the solver with generated skills. While this keeps the solver unchanged and simplifies optimization, it also introduces training latency due to the need for inference-time rollouts. Although the computational cost of parameter updates is relatively low, overall training time is still bounded by solver execution speed, which may limit scalability in environments with expensive or slow simulators.

Third, TARO is currently evaluated in simulated benchmark environments (WebShop and ALFWorld), which, while widely used and standardized, do not fully capture the complexity, noise, and partial observability of real-world settings. As a result, the generalization of TARO to more open-ended or high-stakes interactive environments remains an open question and requires further investigation.

Fourth, the flattened objective is an empirical, biased surrogate; neither its likelihood anchor nor its sampled reference penalty provides an unbiased correction for adaptive refinement data. A local SkillOS reproduction is unavailable; published numbers do not establish a matched-protocol comparison.

References

Salaheddin Alzubi, Noah Provenzano, Jaydon Bingham, Weiyuan Chen, and Tu Vu. 2026. Evoskill: Automated skill discovery for multi-agent systems.

Anthropic. 2024. The claude 3 model family: Opus, sonnet, haiku.

Anthropic. 2025. Agent skills. Claude API Docs. Accessed: 2026-04-01.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, and 1 others. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*.

Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Baobao Chang, and 1 others. 2024. A survey on in-context learning. In *Proceedings of the 2024 conference on empirical methods in natural language processing*, pages 1107–1128.

Google. 2025. Introducing the gemini 2.5 computer use model.

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. 2023. Self-Refine: Iterative refinement with self-feedback. *Preprint*, arXiv:2303.17651.

OpenAI. 2025. Openai computer-using agent.

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

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *Preprint*, arXiv:1707.06347.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *Preprint*, arXiv:2402.03300.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew J. Hausknecht. 2021. Alfworld: Aligning text and embodied environments for interactive learning. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Songjun Tu, Chengdong Xu, Qichao Zhang, Yaocheng Zhang, Xiangyuan Lan, Linjing Li, and Dongbin Zhao. 2026. Dynamic dual-granularity skill bank for agentic rl. *ArXiv preprint*, abs/2603.28716.

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

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. 2024a. A survey on large language model based autonomous agents. *Frontiers Comput. Sci.*, 18(6):186345.

Zora Zhiruo Wang, Apurva Gandhi, Graham Neubig, and Daniel Fried. 2025. Inducing programmatic skills for agentic tasks. In *Second Conference on Language Modeling*.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. 2024b. Agent workflow memory. *arXiv preprint arXiv:2409.07429*.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. 2023. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. *Preprint*, arXiv:2308.08155.

Rong Wu, Xiaoman Wang, Jianbiao Mei, Pinlong Cai, Daocheng Fu, Cheng Yang, Licheng Wen, Xuemeng Yang, Yufan Shen, Yuxin Wang, and 1 others. 2025a. Evolver: Self-evolving llm agents through an experience-driven lifecycle. *arXiv preprint arXiv:2510.16079*.

Yaxiong Wu, Sheng Liang, Chen Zhang, Yichao Wang, Yongyue Zhang, Huifeng Guo, Ruiming Tang, and Yong Liu. 2025b. From human memory to ai memory: A survey on memory mechanisms in the era of llms. *ArXiv preprint*, abs/2504.15965.

Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, and 1 others. 2026. Skillrl: Evolving agents via recursive skill-augmented reinforcement learning. *ArXiv preprint*, abs/2602.08234.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao,

Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

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

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022a. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2022b. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*.

Chunhui Zhang, Yiren Jian, Zhongyu Ouyang, and Soroush Vosoughi. 2024. Working memory identifies reasoning limits in language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 16896–16922.

Shengtao Zhang, Jiaqian Wang, Ruiwen Zhou, Junwei Liao, Yuchen Feng, Zhuo Li, Yujie Zheng, Weinan Zhang, Ying Wen, Zhiyu Li, and 1 others. 2026. Memrl: Self-evolving agents via runtime reinforcement learning on episodic memory. *ArXiv preprint*, abs/2601.03192.

Longtao Zheng, Rundong Wang, Xinrun Wang, and Bo An. 2024. Synapse: Trajectory-as-exemplar prompting with memory for computer control. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

A Additional Experimental Details

A.1 Implementation and Reproducibility Details

This section consolidates the implementation and evaluation protocol used by TARO. We report only settings supported by the experimental configuration and omit unavailable run-specific metadata.

Models. The trainable skill generator π_θ is initialized from Qwen3-8B-Instruct. The solver π_s is frozen throughout training and evaluation; no solver parameters receive policy-gradient updates. We evaluate generated skills with frozen Qwen3-8B, Qwen3-32B, and Gemini-2.5-Pro solvers. Gemini-2.5-Pro is accessed through the OpenRouter Chat Completions API using the model string `google/gemini-2.5-pro`. A generated skill is prepended to the solver prompt as procedural guidance, while the solver continues to select the atomic environment actions.

Trajectory-to-feedback parser. An external API model is used only as a trajectory-to-feedback parser. After an unsuccessful rollout, it maps the task, observed trajectory, and, when applicable, the failed skill and previous feedback into structured textual feedback. It does not generate skills, select environment actions, assign rewards, or receive policy-gradient updates. The implementation uses the DeepSeek-compatible Chat Completions endpoint with model string `deepseek-v4-pro`.

The parser prompt requests a strict JSON object containing an abstracted trajectory, one failure type, the observed bad-action pattern, the missed opportunity, hard constraints, a recommended correction, an avoidance rule, and a concise failure reason. Parsing uses temperature 0, a maximum of 512 output tokens, and no provider-side structured-output mode. Each request permits up to three attempts with a 180-second timeout per attempt and retry delays of 0.5 and 1.0 seconds. The parser removes reasoning blocks and Markdown fences before extracting the first JSON object. Missing fields are repaired deterministically; malformed responses fall back to a rule-based trajectory diagnosis. If the API backend fails, the implementation first attempts local-model parsing and then uses the same rule-based fallback. The complete WebShop parser prompt is provided in Appendix A.7.

Reward. For a task x and generated skill z, the frozen solver returns the binary terminal reward

$$R(x, z) = \begin{cases} 1, & \text{if the solver successfully completes } x, \\ 0, & \text{otherwise.} \end{cases} \tag{19}$$

The feedback parser neither assigns nor modifies this reward. WebShop's continuous environment score is retained only as rollout metadata and an evaluation metric; it does not contribute to generator optimization. Timeouts, invalid actions, and episodes that reach the environment-step limit receive reward 0 unless the task has already reached terminal success. ALFWorld and WebShop success rates are computed from the same binary success indicator.

Skill sampling and refinement. For every training task, the solver first performs one unguided rollout. Tasks solved without a skill are excluded from generator optimization. For each remaining initial failure context C_0, the generator samples $G = 3$ initial skills,

$$z_i^{(1)} \sim q_b(\cdot \mid C_0), \qquad i = 1, \ldots, G. \tag{20}$$

Every branch executes at most $K = 3$ skill variants, including its initial skill, and stops as soon as its current skill succeeds. A successful training-time skill is eligible for insertion into the skill bank. Hence, a failed training context produces a variable-size flattened group containing at most $GK = 9$ executed skill variants, in addition to the shared unguided rollout. An unexecuted candidate receives no reward and is excluded from both GRPO and the skill bank.

Initial and refined skills use the same actor and decoding configuration; refinement changes only the textual context. Skills are sampled with temperature 0.8, top-$p = 1.0$, top-k disabled, and a maximum of 2,048 generated tokens. Generator prompts are limited to 4,096 tokens and are left-truncated when necessary. Complete `<think>` blocks are removed before execution. Empty output supplies no additional solver guidance. The reflective path does not reject a candidate solely for failing a separate structural skill parser, and duplicate sampled strings are not removed before execution.

Retrospective flattening and the clipped surrogate. For task x, let

$$\mathcal{V}_x = \left\{ \left(z_i^{(k)}, r_i^{(k)} \right) \mid i = 1, \ldots, G,\ k = 1, \ldots, K_i \right\} \tag{21}$$

contain all executed variants and their binary rewards. TARO re-associates every executed variant with the same initial context C_0. The implementation retokenizes each revised skill under C_0, discards rollout-time log probabilities computed under its chronological refinement prompt, and recomputes fixed pre-update anchor log probabilities under the flattened context. This re-evaluation changes the loss context without changing where the skill was sampled. Each variant is assigned its immediate candidate reward rather than the cumulative reward of its branch.

The group-relative advantage is

$$A_i^{(k)} = \frac{r_i^{(k)} - \mu_x}{\sigma_x + 10^{-6}}, \tag{22}$$

where μ_x and σ_x are computed over the executed variants in $\mathcal{V}_x$. Constant-reward groups are not explicitly discarded: their centered policy advantage is zero, while their valid response tokens may still contribute to the empirical reference-policy penalty.

For response token t, the anchored likelihood ratio is

$$\rho_{i,t}^{(k)}(\theta) = \exp\left[\log \pi_\theta(y_{i,t}^{(k)} \mid C_0, y_{i,<t}^{(k)}) - \log \pi_{\text{old}}(y_{i,t}^{(k)} \mid C_0, y_{i,<t}^{(k)}) \right]. \tag{23}$$

Clipping is applied independently to token-level ratios, and only non-padding response tokens enter the loss through the response mask. The implementation does not form a complete-sequence or explicitly length-normalized sequence ratio.

A distinct reference policy π_{ref} supplies an empirical reference penalty with coefficient $\beta_{\text{KL}} = 0.01$. The recorded implementation uses the nonnegative token-level quantity

$$d_{i,t}^{(k)} = \exp(\Delta_t) - \Delta_t - 1, \qquad \Delta_t = \log \pi_{\text{ref},t} - \log \pi_{\theta,t}. \tag{24}$$

The old-policy log probabilities used in Equation 23 are recomputed after each rollout batch and immediately before its actor update. They represent a fixed pre-update generator snapshot evaluated under C_0, not the behavior likelihoods of later refinements. Both policies in the reference penalty are likewise evaluated under C_0 and the same response prefix. Since those tokens and prefixes come from the refinement collection process, the empirical penalty is not asserted to be an unbiased estimate of the sequence-level KL under the current policy.

Optimization. We use a learning rate of 1×10^{-6}, a reported optimization batch size of 128, and 16 gradient-accumulation steps. The actor is optimized with AdamW under FSDP. Gradient checkpointing and gradient-norm clipping are enabled, and model computation uses bfloat16 mixed precision with FP32 reductions. Checkpoints are saved periodically and at the final optimization step. Training-time validation and automatic early stopping are disabled; the reported results use the designated training checkpoint rather than an automatically selected validation optimum.

Skill bank and retrieval. A separate skill bank is initialized empty for each benchmark and stores successful training-task skills. Retrieval uses Qwen3-Embedding-0.6B through SentenceTransformers. Each entry is represented by the concatenation of its task and skill text. The current task is compared with all entries using cosine similarity; the ten most similar entries are retained, reranked by descending reuse count, and the top two are inserted into the generator context. Embeddings and ranking are computed in memory without a separate vector-index service.

Exact duplicates are defined by equality of both task text and skill text. Reinserting a duplicate increments its reuse count; a distinct skill is appended with count zero. Stored metadata include the task, skill, goal index, success round, binary reward, and source. The bank has no fixed capacity or eviction policy, so multiple distinct successful skills from one task may be retained. Held-out evaluation freezes the bank and permits only skills collected from seen training tasks; skills produced on held-out examples are never written back.

Datasets and evaluation protocol. WebShop uses the standard 10,587/1,000/500 train/development/test split. The category-held-out experiment applies the leave-one-category-out protocol described in Appendix A.2 within these split boundaries. The reported ALFWorld evaluation contains 140 tasks: 35 Pick, 13 Look, 27 Clean, 16 Heat, 25 Cool, and 24 Pick2 tasks. For every evaluation task, the frozen solver first performs an unguided rollout. If it succeeds, that result is retained and no assisted retry is run. TARO does not perform multi-round test-time refinement in the standard setting. It uses one parsed failed rollout to generate a single skill-conditioned retry. Iterative comparison variants may parse later failed skill-conditioned rollouts and execute up to three skills, stopping at the first success. Conditioned on an initial failure, standard evaluation uses one parser call and one skill-conditioned solver episode; iterative evaluation uses between one and three of each.

WebShop episodes are limited to 14 environment actions and retain the two most recent observation–action pairs in the prompt history. ALFWorld episodes are limited to 50 actions and use the same history length. Local Qwen solver evaluation uses temperature 0.7, top-$p = 1.0$, a maximum of 1,024 new tokens per action, and sampling enabled. Generation stops on `</action>` or `<|im_end|>`. The solver prompt requires one reasoning span in `<think>` tags and one action in `<action>` tags; the parsed action is projected to the environment action space. Empty or failed generations are converted to a no-op action and receive no success reward.

Final success rate and mean WebShop score are arithmetic means over all evaluation tasks, combining retained unguided successes with assisted results for initially failed tasks. We report ALFWorld SR and both WebShop score and SR.

For ALFWorld, let $\hat{p}_j$ denote the empirical final-success frequency for task j. For category c with n_c tasks, task-level aggregation is

$$\mathrm{SR}_c = \frac{100}{n_c} \sum_{j \in c} \hat{p}_j, \qquad \mathrm{Avg.} = \frac{\sum_c n_c \, \mathrm{SR}_c}{140}, \tag{25}$$

where $(n_c) = (35, 13, 27, 16, 25, 24)$ in the order used in Table 1. Thus Avg. is weighted by task counts, rather than taking an unweighted mean of the six categories.

Software and numerical configuration. The implementation uses Python 3.11, a VERL-derived GRPO trainer, vLLM for local batched generation, FSDP for distributed optimization, and the vendored ALFWorld 0.4.2 environment. Generator-side gradient checkpointing is enabled. Model parameters use bfloat16 mixed precision with FP32 reduction and buffer accumulation; no quantization or DeepSpeed/ZeRO configuration is used. Experiments run on eight 80-GB GPUs. Environment and WebShop shuffle seeds are set to 0. External API responses remain a known nondeterministic component.

A.2 Held-Out Splits and Leakage Controls

WebShop split. We use the standard split of 10,587 training, 1,000 development, and 500 test tasks. Category membership is taken directly from the coarse `category` field in each WebShop goal record. The five categories are `beauty`, `electronics`, `fashion`, `garden`, and `grocery`.

We conduct five-fold leave-one-category-out evaluation. In each fold, one category is held out and the remaining four are treated as seen; every category serves as the held-out category exactly once. Training uses seen-category tasks from the training split, development uses seen-category tasks from the development split, and held-out evaluation uses the held-out category's tasks from the test split. The standard train/development/test boundaries are preserved in every fold. Held-out-category tasks are excluded from policy optimization, refinement-data collection, and skill-bank construction. We sample a size-matched seen evaluation set from the test split within each fold and stratify it by the number of product constraints to reduce difficulty imbalance.

ALFWorld split. We hold out task–object–receptacle/room combinations. Individual task templates, objects, and receptacles may occur during training, but their held-out combinations do not. This tests transfer to new procedural compositions without introducing an unseen action space.

Leakage controls. Splits are fixed before training. During held-out evaluation, the skill bank is frozen and contains only skills collected from seen tasks. Skills produced on held-out tasks are neither written to the bank nor made available to later examples. All methods use the same split, frozen solver, episode limits, initial rollout, feedback parser, prompts, and decoding settings. The reflection baseline uses the initial generator with the same execution–feedback–revision protocol at evaluation, but receives no TARO policy updates and performs no retrospective flattening. TARO is evaluated in its one-shot setting.

The complete seen/held-out results are reported in Table 2 in the main experimental section.

A.3 Refinement Depth and Adaptive Stopping

Refinement depth. We vary the maximum number of executed skill variants per branch over $K \in \{1, 2, 3, 4\}$. Thus, $K = 1$ denotes one-shot skill generation rather than a no-skill condition. Efficiency is the gain in SR percentage points per 100 additional seconds per parameter update relative to the preceding row:

$$\text{Eff.}(K) = 100\,\frac{\text{SR}(K) - \text{SR}(K-1)}{\text{Time}(K) - \text{Time}(K-1)}. \tag{26}$$

Table 5 in the main text reports all depth, time, and efficiency measurements. In both Tables 5 and 6, Time is the measured time in seconds per parameter update. The efficiency metric therefore relates final evaluation SR gains to additional time per parameter update.

Adaptive stopping. The standard TARO loop stops a branch when its current skill succeeds. To avoid executing nearly unchanged revisions on unsuccessful branches, we additionally use a text-based stagnation rule that adds no judge model, embedding model, or API call. Let $\mathcal{T}(z)$ be the token sequence obtained after collapsing repeated whitespace in skill z and applying the Qwen tokenizer without special tokens. For consecutive variants, define

$$D\Big(z^{(k+1)}, z^{(k)}\Big) = \frac{\text{Lev}\big(\mathcal{T}(z^{(k+1)}), \mathcal{T}(z^{(k)})\big)}{\max\big(|\mathcal{T}(z^{(k+1)})|, |\mathcal{T}(z^{(k)})|\big)}, \tag{27}$$

where Lev is token-level Levenshtein distance. We fix $\delta = 0.05$ before evaluation. After an unsuccessful execution of $z^{(k)}$, TARO generates $z^{(k+1)}$. If $D(z^{(k+1)}, z^{(k)}) < \delta$, the branch stops before executing the near-duplicate. Because that candidate has no environment reward, it is excluded from the flattened GRPO group and the skill bank. Otherwise, refinement continues until success or $K_{\max} = 3$ executed variants.

The complete performance and cost comparison is in Table 6. Average executed skill rollouts are computed per training branch, including the initial skill and excluding unexecuted revisions and the shared unguided rollout. Both variants use the same generator initialization, training tasks, frozen solver, feedback parser, prompts, decoding settings, $G = 3$, and $K_{\max} = 3$.

A.4 Attribution-Control Definitions

The trajectory-to-feedback parser converts an observed failed rollout into a compact diagnosis containing the failed action pattern, missed opportunity, hard constraints, and a proposed behavioral correction. It does not generate the revised skill. We use the following controls:

- **Full TARO** uses trajectory-specific parsed feedback, temporal refinement, retrospective flattening, and the clipped surrogate.
- **Without retrospective flattening** reuses exactly the same collected refinement data and parser outputs as Full TARO, but optimizes each revised skill under its chronological refinement context rather than re-associating it with C_0.
- **Without temporal refinement** retains the parsed diagnosis of the initial failure but samples and optimizes only a one-shot group. No refined variants exist, so flattening is not applicable.
- **Generic feedback** replaces the trajectory-specific diagnosis with the fixed message "the previous attempt failed; revise the skill," while retaining refinement, flattening, and the clipped surrogate.

Table 9: Generated-skill quality on 100 paired held-out tasks. Entries report positive adjudicated labels over evaluated tasks, followed by percentages. Δ is TARO minus Base Generator in percentage points.

Dataset	Generator	Task-relevant	Executable	Specific enough	Non-hallucinated	Overall useful
WebShop ($n = 50$)	Base Generator	39/50 (78)	43/50 (86)	28/50 (56)	34/50 (68)	30/50 (60)
	TARO	**44/50 (88)**	**45/50 (90)**	**39/50 (78)**	**43/50 (86)**	**39/50 (78)**
	Δ (pp)	+10	+4	+22	+18	+18
ALFWorld ($n = 50$)	Base Generator	40/50 (80)	44/50 (88)	31/50 (62)	37/50 (74)	32/50 (64)
	TARO	**45/50 (90)**	**46/50 (92)**	**40/50 (80)**	**44/50 (88)**	**41/50 (82)**
	Δ (pp)	+10	+4	+18	+14	+18
Overall ($n = 100$)	Base Generator	79/100 (79)	87/100 (87)	59/100 (59)	71/100 (71)	62/100 (62)
	TARO	**89/100 (89)**	**91/100 (91)**	**79/100 (79)**	**87/100 (87)**	**80/100 (80)**
	Δ (pp)	+10	+4	+20	+16	+18

- **Final reward only** removes textual failure feedback and refinement, reducing training to GRPO over initially sampled skills with binary terminal rewards.

All variants use the same task split, generator and solver checkpoints, and clipping and advantage-normalization settings. The main-table GRPO baseline uses a training skill-rollout budget aligned with TARO. The primary attribution comparison is Full TARO versus the variant without retrospective flattening. Because these two settings use exactly the same collected refinement data and parser outputs, their difference isolates training-context assignment within this fixed-data comparison, rather than a change in the collected skill variants. Generic feedback separately measures the value of trajectory-specific parsing, so the parser's contribution is reported rather than hidden or treated as part of the learned policy.

A.5 Internalization Protocol and Cost Accounting

The Base Generator is the Qwen3-8B-Instruct checkpoint used to initialize π_θ before policy optimization. TARO uses the same architecture after training. The values in Table 4 and the seen/held-out analysis come from the same split-controlled evaluation, whereas Table 1 reports the separate full evaluation set; this accounts for the different Base Generator values. For every evaluation task, the frozen solver first attempts the task without generated guidance. If this initial rollout fails, the feedback parser converts it into the common initial context used by all variants in Table 4.

The one-shot variants generate exactly one skill and execute one skill-conditioned solver episode. Test-time-refinement variants parse a failed skill-guided rollout, revise the skill, and try again, up to three executed skills with early stopping on success. Conditioned on an initial failure, one-shot evaluation therefore uses one parser call and one skill-conditioned rollout, whereas test-time refinement uses one to three parser calls and one to three skill-conditioned rollouts. The comparison between TARO one-shot and the Base Generator with test-time refinement tests whether training internalizes repeated, execution-grounded correction into a single generated skill. Comparing TARO one-shot with TARO plus test-time refinement measures the remaining benefit of additional evaluation-time interaction.

A.6 Skill-Quality Annotation Protocol

We sample 100 held-out tasks, with 50 from WebShop and 50 from ALFWorld. For every task, we generate one skill with Base Generator and one with TARO under the same initial context, prompt, decoding settings, and output-length limit. The 200 skills are anonymized, randomly ordered, and independently labeled by two annotators; disagreements are resolved by adjudication.

Annotators assign binary labels for (1) task relevance, (2) executability under the environment action space, (3) sufficient specificity, (4) non-hallucination, and (5) overall usefulness. A conditional instruction is counted as hallucinated only when it presents an unobserved condition as fact.

The largest overall gains are in specificity (+20 points), usefulness (+18), and non-hallucination (+16), while executability remains high for both generators.

A.7 Prompt Templates

The following templates document the verified WebShop prompts and the verified ALFWorld solver prompt. In the standard TARO setting, an unguided failure is parsed once, the generator produces one skill from that diagnosis, and the frozen solver performs one skill-conditioned retry. Blocks for a previous failed skill or previous diagnosis are used only during training-time refinement or the explicitly labeled iterative evaluation variants.

A.7.1 WebShop Solver Execution Prompt

The frozen WebShop solver receives no generated skill during the initial rollout. After an initial failure, the generated task-specific skill is inserted immediately before the current-progress block for the standard single retry; iterative variants reuse the same insertion point.

Prompt A.1: WebShop Solver Execution Prompt

System Prompt:
You are an expert autonomous agent operating in the WebShop e-commerce environment.
Your task is to: `{task_description}`

[Optional Skill Injection: Inserted only during reflective retries after initial failure]
Task-Specific Skill To Follow
`{generator_generated_skill}`

Current Progress

Prior to this step, you have already taken `{step_count}` step(s). Below are the most recent `{history_length}` observations and the corresponding actions you took: `{action_history}`

You are now at step `{current_step}` and your current observation is: `{current_observation}`
Your admissible actions of the current situation are: [`{available_actions}`].

Now it's your turn to take an action. You should first reason step-by-step about the current situation. This reasoning process **MUST** be enclosed within `<think> </think>` tags. Once you've finished your reasoning, you should choose an admissible action for current step and present it within `<action> </action>` tags.

A.7.2 WebShop Failure Diagnosis Prompt

The following prompt is sent to an API model to summarize a failed solver trajectory. The output is a structured behavior-only diagnosis that identifies the failed action pattern, missed opportunity, hard constraints, and an explicit avoidance rule. This diagnosis becomes the main conditioning signal for the skill generator.

Prompt A.2: WebShop Failure Diagnosis via API Summary

You are analyzing a failed WebShop agent rollout.
[TASK]
`{task}`
[SKILL USED]
`{skill_text}`

[PREVIOUS FAILURE REASON]
`{previous_failure_reason}`

[COMPACT TRAJECTORY]
`{compact_trajectory}`

[RAW SOLVER THOUGHT/ACTION TRACE]
`{raw_thought_action_trace}`

[FINAL TASK SCORE]
`{final_score}`

Write a strict JSON object with exactly these keys:
- `"compact_trajectory"`: concise search/click chain using tokens like `search[...]`, `click[item]`, `click[back to search]`, `click[option]`, `click[buy now]`
- `"failure_type"`: exactly one label from this taxonomy: `search_loop`, `pagination_loop`, `no_candidate_inspection`, `wrong_product_type`, `missing_variant_selection`, `constraint_not_verified`, `price_violation`, `premature_buy`, `navigation_dead_end`, `malformed_or_noop_actions`, `insufficient_search_query`, `overconstrained_search_query`, `other`
- `"bad_action_pattern"`: what the solver actually did wrong in this rollout
- `"missed_opportunity"`: what the solver should have done instead at that point
- `"constraint_focus"`: array of hard task constraints that must be preserved, such as size, color, price cap, material, and gender
- `"recommended_fix"`: one concrete behavior change for the next attempt
- `"avoid_rule"`: one explicit behavior to avoid repeating
- `"failure_reason"`: one concise sentence summarizing the diagnosed failure

Diagnosis rules:
- Do **NOT** blame max step limit.
- Do **NOT** claim the task is impossible.
- Use the raw thought/action trace to diagnose the solver's reasoning and chosen actions.
- Do **NOT** infer page contents that are not present in the trace.
- Prefer specific behavior patterns such as repeated search, repeated pagination, no product inspection, wrong category, missing size/color selection, buying too early, or failing to verify hidden constraints.
- If the trajectory shows `search[...]` followed by many `click[next >]`, prefer `failure_type=pagination_loop`.
- If the trajectory shows repeated `search[...]` without `click[item]`, prefer `failure_type=search_loop` or `failure_type=no_candidate_inspection`.
- Output JSON only, no markdown and no extra text.

Optional fields: The **[SKILL USED]** and **[PREVIOUS FAILURE REASON]** blocks are included only in reflective rounds after an earlier generated skill has failed.

A.7.3 WebShop Skill Generator Prompts

The following prompts are used to generate a new task-specific skill from the failed trajectory diagnosis. The default format is a compact *Strategy/Notes* block. We also support a *Goal/Do/Avoid* variant for ablation.

A.7.4 ALFWorld Solver Execution Prompt

The following skill-conditioned ALFWorld solver template is reproduced verbatim from the evaluated implementation. The auxiliary ALFWorld trajectory-parser and generator templates are not reproduced because they were not available as complete versioned artifacts in the accompanying source snapshot.

Prompt A.3.1: Reflective Skill Generator, Task-Specific Style

You are a Skill Generator for a WebShop solver agent.
Write a concise task-specific skill that helps the frozen solver complete the task.

[CURRENT TASK]
`{task}`

[FAILED SOLVER TRAJECTORY]
`{compact_trajectory}`

[STRUCTURED FAILURE DIAGNOSIS]
Type: `{failure_type}`
Bad action pattern: `{bad_action_pattern}`
Missed opportunity: `{missed_opportunity}`
Recommended fix: `{recommended_fix}`
Avoid rule: `{avoid_rule}`
Hard constraints to preserve: `{constraint_focus}`
Summary: `{failure_reason}`

[PREVIOUS SKILL THAT FAILED]
`{previous_skill}`

Revise the previous skill. Do not repeat it. Fix the behavior described in the structured diagnosis.

[REFERENCE SKILLS FROM MEMORY BANK]
`{retrieved_skills}`

[OUTPUT]
Return exactly one task-specific markdown block. Do not output a Retrieved Relevant Experience/RPE/RRE block.

Task-Specific Skill

Strategy
- Start with a short concrete WebShop search query for THIS task when search is useful. Use 3–6 keywords and do not include every constraint.
- Explain when to inspect a product instead of repeating the failed action pattern.
- Explain how to verify THIS task's hard constraints before buying.

Notes
- Key constraints: list the concrete constraints from the task.
- Avoid: name the specific failed behavior from the diagnosis and how not to repeat it.

Rules: no `<think>`/`<action>` tags, code fences, angle-bracket placeholders, generic shopping recipes, or RPE/RRE headings. Every line must be specific to this task or the diagnosis.

Optional fields: The **[PREVIOUS SKILL THAT FAILED]** block appears only after the first reflective round. If no memory bank is enabled, `{retrieved_skills}` is set to `(none)`.

Prompt A.3.2: Reflective Skill Generator, Goal/Do/Avoid Style

The prompt body is the same as Prompt A.3.1 through the **[REFERENCE SKILLS FROM MEMORY BANK]** block. The output specification is replaced by the following format.

[OUTPUT FORMAT]
Output ONLY one markdown block. Do not include `<think>`, `<action>`, code fences, commentary, RPE/RRE headings, or angle-bracket placeholders.

Task-Specific Skill

Goal
State the exact item and hard constraints for this task in one sentence.

Do
- Give one concrete WebShop search query with 3–6 keywords only. Do **NOT** include every constraint in the search query.
- Give one rule for clicking and inspecting a plausible product from the first non-empty results page before paginating or searching again.
- Give one rule for verifying the remaining task-specific constraints on the product page before buying.

Avoid
- Do not repeat the same or similar search query after results appear.
- Do not click next > before inspecting at least one visible candidate.
- Do not go back to search unless the inspected product clearly violates a hard constraint.
- Name the specific failed behavior from the trajectory and how to avoid repeating it.

Prompt A.4: ALFWorld Solver Execution Prompt

You are an expert agent operating in the ALFRED Embodied Environment. Your task is to: `{task_description}`.

skills
`{skills}`

Current Progress
Prior to this step, you have already taken `{step_count}` step(s). Below are the most recent `{history_length}` observations and the corresponding actions you took: `{action_history}`
You are now at step `{current_step}` and your current observation is: `{current_observation}`
Your admissible actions of the current situation are: [`{admissible_actions}`].

Now it is your turn to take one action for the current step. You should first reason step-by-step about the current situation, then think carefully which admissible action best advances the task goal. This reasoning process **MUST** be enclosed within `<think> </think>` tags. Once you have finished your reasoning, choose one admissible action and present it within `<action> </action>` tags.

A.8 Source and Protocol of Published SkillOS Results

The SkillOS rows in Table 1 are transcribed from Tables 1–2 of the original paper (Ouyang et al., 2026). We retain the reported values for all six ALFWorld categories and aggregate SR, as well as WebShop score and SR, without re-aggregating them under TARO's metric definition.

SkillOS uses a trained Qwen3-8B curator with each frozen solver. Its evaluation uses 140 ALFWorld `valid_seen` tasks and 500 WebShop test tasks, with a streaming SkillRepo updated across tasks, BM25 top-5 retrieval, and a 30-action limit. TARO includes a failure-conditioned retry and action limits of 50/14 for ALFWorld/WebShop. These differences mean that the published rows provide a cross-paper comparison rather than a local reproduction under a matched protocol.

A.9 Training-Budget Comparison and Reporting Scope

Existing GRPO comparison. The GRPO results in Table 1 use the same training skill-rollout budget as TARO, measured by frozen-solver executions for candidate-skill evaluation. GRPO uses independent candidates without sequential refinement.

TARO samples $G = 3$ initial skills and executes at most $K = 3$ variants per branch, including the initial candidate. Thus each initial failed task allows at most $GK = 9$ skill-conditioned solver executions, in addition to the shared unguided attempt. Successful branches stop early.

Fixed-data flattening comparison. Full TARO and Without RF reuse exactly the same collected refinement data and parser outputs. Full TARO pairs the skill variants with C_0; Without RF retains their chronological refinement contexts. This comparison controls the collected data when testing training-context assignment. It does not by itself measure the benefit of sequential refinement relative to independent sampling.

Local SkillOS comparison. A local adaptation should retain SkillOS's learned repository curation and identify protocol changes explicitly. Use the same solver, task IDs, action limits, and evaluation interaction budget. For a frozen-bank transfer evaluation, curate only on training tasks and disable test-task writeback for both methods; label this as a frozen-bank adaptation, because original SkillOS is streaming. A separate streaming comparison must expose both methods to the same task order and repository update rules. Count curation training and evaluation costs, and distinguish the method's reward design from shared environmental task-success metrics.