POLICY-RELATIVE FRONTIER LEARNING FOR SELF-EVOLVING LANGUAGE AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

As language agents are deployed across increasingly diverse tasks, self-evolution offers a practical way to adapt them through their own experience under limited budgets and strict data privacy requirements. Yet many real-world domains provide only outcome feedback, leaving the policy's current capability bottlenecks unclear. Each round of self-evolution therefore raises a central question: what should the policy learn next, and how far should supervision extend? We propose **Policy-Relative Frontier Learning**, a framework that uses progress differences among the policy's own rollouts to determine the boundaries and granularity of supervision. Using the shortest successful rollout as a reference, we identify the furthest shared progress of each failed rollout as its relative frontier. Each failed rollout learns from its own frontier to the next frontier in the group, or to task completion if no further frontier exists. By focusing on local progress, this design recognizes that even a rollout that ultimately fails can contain useful segments that help another attempt cross its relative frontier. We therefore also select segments from failed rollouts, expanding supervision beyond successful trajectories. We fine-tune the policy on the selected segments and reconstruct learning targets from fresh experience after each update. Experiments on ALFWorld and WebShop show that our method outperforms baselines with fewer rounds of evolution. Ablations reveal that the effectiveness of fixed supervision granularity varies across tasks, highlighting the value of policy-relative learning intervals derived from the policy's own progress. Code is available at `https://anonymous.4open.science/r/relative-frontier-evolve-657F/`.

1 INTRODUCTION

Adapting LLM agents to specialized domains often requires learning new tasks from limited high-quality training data. Obtaining demonstrations or step-level supervision from human annotators or stronger models can make this adaptation costly (Tao et al., 2024; Yuan et al., 2025). Self-evolution offers a way to reduce this dependence by allowing agents to collect experience through their own interactions and use environmental feedback to update their policies (Tao et al., 2024; Dou et al., 2024; Yuan et al., 2025). However, without explicit correctness labels for intermediate decisions, agents lack fine-grained guidance for learning from this experience. Process-supervised reward modeling demonstrates the value of such guidance in mathematical reasoning (Lightman et al., 2024). This raises a central challenge: how can we construct effective training targets when only outcome supervision is available?

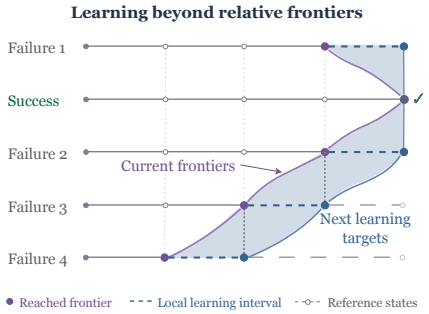


Figure 1: Relative frontiers define local learning intervals for failed rollouts.

Existing methods use outcome feedback either to reinforce sampled behavior through returns or advantages, as in GRPO, GiGPO, and SALT (Shao et al., 2024; Feng et al., 2025; Li et al., 2026), or to construct explicit targets for supervised self-training. A common approach is to train on complete, outcome-verified solutions, obtained through successful-sample selection or additional ratio-

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.

nalization and reflection, as exemplified by RFT, STaR, ReST$^{\mathrm{EM}}$, and Re-ReST (Yuan et al., 2023; Zelikman et al., 2022; Singh et al., 2024; Dou et al., 2024). These approaches provide a simple and effective foundation for self-training. Yet failed trajectories also contain valuable learning information: prior work shows that incorporating appropriately processed unsuccessful trajectories can improve fine-tuning over training on successful examples alone (Wang et al., 2024). Failure-based revision further demonstrates that these trajectories can help identify where the current policy needs correction and support the construction of targeted training examples (Yuan et al., 2025; Lin et al., 2026). Together, these findings motivate us to examine failed rollouts as evidence of the boundary between progress already demonstrated and progress that remains to be learned.

Comparing successful and failed rollouts on the same task reveals shared progress. Their differences help identify where additional supervision is needed. Prior work uses such comparisons to construct corrective targets. Agent-R connects failed trajectories to successful neighboring branches, while UI-Voyager transfers successful actions at matched states into failed contexts (Yuan et al., 2025; Lin et al., 2026). Yet how much should a failed attempt learn? A single corrective action may not capture a transition requiring several dependent decisions. A complete successful continuation, in contrast, may extend beyond the immediate learning need, reducing supervision efficiency.

To address the challenge of constructing learning targets that match the current policy's capabilities on each task, we propose **Policy-Relative Frontier Learning**, which uses progress differences among the policy's own rollouts to determine what to learn and how far supervision should extend. For each task, we collect a group of rollouts and use the shortest successful trajectory as a reference. The furthest shared progress between each failed rollout and this reference defines its *relative frontier*. As illustrated in Figure 1, each failed rollout learns from its own frontier to the next further frontier within the group, or to task completion when no further failed frontier exists. This defines its *relative learning interval*. We supervise each relative learning interval using a segment from another rollout in the group. This allows a failed rollout to learn from either a successful attempt or another failure that has made further progress. Since successful rollouts can contain redundant actions or detours, failed rollouts may offer more direct alternatives while broadening the range of learning examples. We retain the failed rollout's prefix as context and fine-tune only on the selected segment. Repeating this process with fresh rollouts reconstructs the learning targets as the policy is updated, yielding an adaptive curriculum. Our contributions are as follows:

- We identify a central challenge in self-evolution: deciding what the current policy should learn and how far supervision should extend. We propose determining such boundaries from relative progress across the policy's own rollouts, without external knowledge on task stages or subgoals.
- We introduce **Policy-Relative Frontier Learning**, which constructs adaptive local learning segments from successful and failed trajectories and rebuilds learning targets after each policy update, enabling iterative self-evolution.
- Experiments on ALFWorld and WebShop show that our method achieves the best final overall performance and surpasses the final results of multiple baselines with only 2/3 and 1/2 as many evolution rounds, respectively, demonstrating our higher self-evolution capability.

2 Related Works

Methods that convert generated experience into learning signals fall into trajectory-level and step-level approaches. Trajectory-level methods select, refine, or compare complete executions for learning (Section 2.1). Step-level methods provide finer-grained supervision either by evaluating intermediate decisions (Section 2.2) or by constructing local learning targets (Section 2.3).

2.1 Trajectory-Level Learning from Complete Executions

A classic approach to self-evolution is to learn from complete, outcome-verified solutions: rejection sampling fine-tuning (RFT) retains correct reasoning paths (Yuan et al., 2023), STaR iteratively bootstraps answer-verified rationales (Zelikman et al., 2022), and ReST and ReST$^{\mathrm{EM}}$ alternate between generating samples and learning from reward-filtered outputs (Gülçehre et al., 2023; Singh et al., 2024). Related approaches expand this recipe through reward-ranked sample selection, as in RAFT (Dong et al., 2023), reflection-based data refinement, as in

Re-ReST (Dou et al., 2024), and preferences between higher- and lower-reward agent trajectories, as in ETO (Song et al., 2024). These methods offer simple and effective ways to improve models through learning from complete generated sequences. However, sequence-level outcomes provide limited information about the contributions of individual steps. Process supervision offers finer-grained guidance: Lightman et al. (2024) show that step-level judgments improve reward modeling over outcome supervision in mathematical reasoning.

2.2 Step-Level Learning through Local Credit Assignment

Local credit assignment evaluates intermediate decisions through three broad routes: process reward modeling, rollout-based credit estimation, and task-specific progress supervision. Process reward modeling methods learn to evaluate intermediate decisions: AgentPRM estimates process rewards from rollout returns (Choudhury, 2025), while PRIME learns implicit process rewards from outcome supervision (Cui et al., 2026). Rollout-based methods derive local credit by comparing sampled executions. VinePPO samples additional continuations from intermediate prefixes to estimate their values (Kazemnejad et al., 2025), whereas GiGPO and SALT aggregate information across shared states or transitions in existing rollouts to obtain step-level learning signals (Feng et al., 2025; Li et al., 2026). Task-specific progress supervision instead exploits structured signals from the environment: BEACON uses milestone detectors grounded in environment-defined goal constraints to construct local rewards and segment-level advantages (Wang et al., 2026). These approaches provide evaluative signals indicating which intermediate decisions should be reinforced.

2.3 Step-Level Learning from Targeted Corrections

A complementary approach constructs explicit behavioral targets, offering a direct way to turn experience into supervision for fine-tuning. Reflection-based approaches such as Reflexion summarize feedback to guide subsequent behavior (Shinn et al., 2023), although models can struggle to locate errors even when they can correct them given their location (Tyen et al., 2024). Trajectory comparisons provide more explicit corrective supervision, with methods differing in its source and scope. IPR evaluates policy-generated alternatives against expert actions using estimated step-level rewards and constructs action-level preferences (Xiong et al., 2024). STeCa identifies suboptimal actions through step-level reward comparisons and uses reflection to generate calibrated trajectories (Wang et al., 2025). Agent-R connects a retained failed prefix to a successful neighboring branch in a search tree at an actor-selected revision point (Yuan et al., 2025), while UI-Voyager transfers successful actions at visually matched states into failed contexts (Lin et al., 2026). These approaches provide corrective targets ranging from individual actions to revised trajectories or successful continuations. When expressed as demonstrations, such targets can be learned through straightforward supervised fine-tuning. Our work focuses on determining the boundaries of these targets from relative progress across rollouts generated by the policy.

3 Problem Setup

We study self-evolution in which a policy learns from its own interactions and terminal outcome feedback, without step-level correctness annotations. We consider an interactive task x with environment E_x and a policy π. We omit the evolution-round index when describing a single rollout. At step t, the environment is in an objective state s_t and presents an observation o_t. The policy observes o_t and produces a response $u_t = (\mathrm{thought}_t, \mathrm{action}_t)$. Executing action_t transitions the environment from s_t to s_{t+1}. Although the underlying state is not visible to the policy, we record the state sequence solely for offline training-data construction. Let τ_i denote the ith rollout, with T_i actions and terminal outcome $y_i \in \{0, 1\}$ returned by the environment. For readability, we omit the rollout index from states, observations, and responses:

$$\tau_i = (s_0, o_0, u_0, \ldots, s_{T_i-1}, o_{T_i-1}, u_{T_i-1}, s_{T_i}, o_{T_i}, y_i). \tag{1}$$

At evolution round r, policy π_r generates a rollout pool $\mathcal{R}_r$. A target-construction procedure converts this experience into a supervised training set $\mathcal{Z}_r$, which is used to update the policy:

$$\mathcal{R}_r \xrightarrow{\text{target construction}} \mathcal{Z}_r, \qquad \pi_{r+1} = \mathrm{Update}(\pi_r, \mathcal{Z}_r). \tag{2}$$

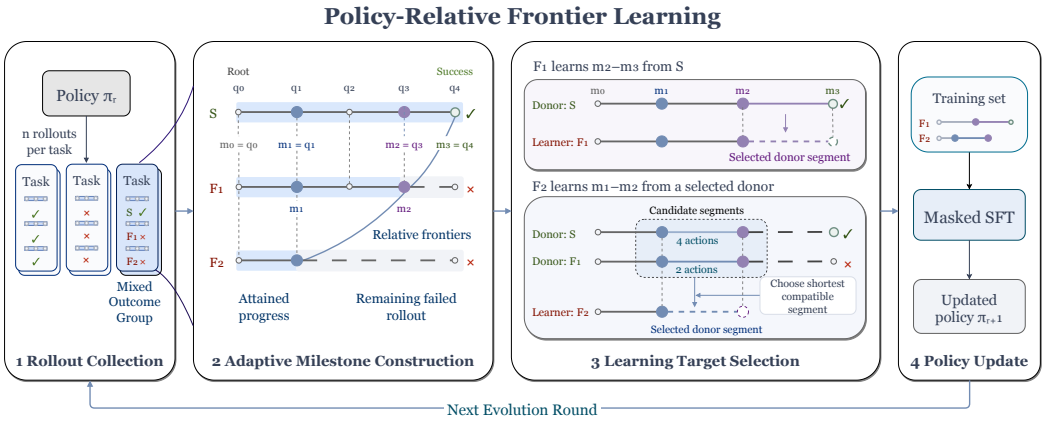


Figure 2: **Overview of Policy-Relative Frontier Learning.** (1) **Rollout Collection:** collect rollouts and retain mixed-outcome groups. (2) **Adaptive Milestone Construction:** identify each failed rollout's relative frontier, then order the distinct frontiers with `Root` and `Success` to form the milestone sequence. (3) **Learning Target Selection:** select compatible segments from successful or farther-progressed failed rollouts for the relative learning intervals. (4) **Policy Update:** retain each failed rollout's prefix as context and fine-tune on the selected segment tokens. The updated policy generates fresh experience for the next round.

Our focus is how to construct learning targets from $\mathcal{R}_r$ that specify both what the policy should learn and how far supervision should extend.

4 Method

An overview of our framework is illustrated in Figure 2. We first sample rollouts for each task using the current round's policy and retain mixed-outcome rollout groups (Section 4.1). Using a successful rollout as a reference, we locate the furthest reference node matched in order by each failed rollout and define it as that rollout's relative frontier. We deduplicate these frontiers and order them along the reference, adding `Root` and `Success` to form a policy-relative milestone sequence for the task (Section 4.2). We retain each failed rollout's prefix as context and select a compatible segment from another rollout to supervise progress from its frontier to the next milestone (Section 4.3). We update the policy through supervised fine-tuning on these segments and repeat the process with fresh rollouts (Section 4.4).

4.1 Rollout Collection and Task Selection

The current policy π generates n rollouts for each task x, forming a rollout group $\mathcal{G}_x = \{\tau_i\}_{i=1}^{n}$. We distinguish all-success groups $\mathcal{G}_x^{\mathrm{s}}$, all-failure groups $\mathcal{G}_x^{\mathrm{f}}$, and mixed-outcome groups $\mathcal{G}_x^{\mathrm{mix}}$. We retain only mixed-outcome groups, which contain both successful rollouts τ_i^{s} and failed rollouts τ_j^{f}. All-failure groups lack a successful reference, while all-success groups may be too easy for the current policy. This selection is consistent with curricula based on the Zone of Proximal Development (ZPD), which prioritize challenging yet learnable examples (Cui & Sachan, 2025).

4.2 Adaptive Milestone Construction

Within each $\mathcal{G}_x^{\mathrm{mix}}$, we select the shortest successful rollout τ_i^{s} as the reference τ^{ref} to reduce unnecessary detours and better capture the steps needed for task completion. We construct reference candidates q_h^{ref} from the recorded states of τ^{ref}, merging consecutive states that are identical under the matching criteria. Here, h indexes candidate positions. For example, in Figure 3, an invalid action leaves the agent on the same search-results page, so these consecutive states yield a single reference candidate. With c internal candidates, the ordered reference sequence is

$$\mathcal{Q}^{\mathrm{ref}} = (q_0^{\mathrm{ref}}, q_1^{\mathrm{ref}}, \ldots, q_{c+1}^{\mathrm{ref}}), \tag{3}$$

where $q_0^{\text{ref}} = \texttt{Root}$ and $q_{c+1}^{\text{ref}} = \texttt{Success}$.

For each failed rollout τ_i^{f}, we construct its candidate sequence $\mathcal{Q}_i^{\text{f}}$ in the same way as $\mathcal{Q}^{\text{ref}}$ and search for reference candidates in order. Starting with the first internal reference candidate q_1^{ref}, we select the earliest candidate in $\mathcal{Q}_i^{\text{f}}$ that describes the same state. As illustrated by τ_3^{f} in Figure 3, the failed rollout may take detours between matching positions, but reference candidates cannot be skipped. After each successful match, we search for the next reference candidate only after the matched position. If no match is found in the remaining sequence, we stop and define the last successfully matched reference candidate as the **relative frontier** of τ_i^{f}. If the first internal reference candidate q_1^{ref} cannot be matched, the frontier is $\texttt{Root}$.

The relative frontiers identify the progress reached by different failed rollouts and provide boundaries for subsequent learning. We deduplicate these frontiers, order them along $\mathcal{Q}^{\text{ref}}$, and include $\texttt{Root}$ and $\texttt{Success}$ to form the milestone sequence

$$\mathcal{M} = (m_0, m_1, \ldots, m_b, m_{b+1}), \tag{4}$$

where b is the number of distinct internal frontiers, $m_0 = \texttt{Root}$, and $m_{b+1} = \texttt{Success}$. Thus, $\mathcal{M}$ is an ordered subsequence of $\mathcal{Q}^{\text{ref}}$, with each m_k denoting a selected reference candidate. These learning boundaries arise from progress demonstrated by the current policy, without manually prescribed task stages or intermediate completion labels.

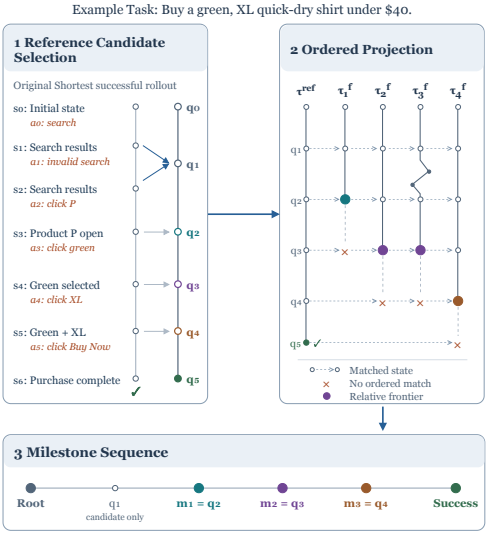


Figure 3: **Adaptive milestone construction.** A simplified WebShop example. Reference candidates q_h^{ref}, indexed by h along the reference, are matched in order against failed rollouts τ_i^{f}. Crosses mark the first unavailable ordered match, and colored nodes mark the last match—the relative frontier. Distinct frontiers define internal milestones; q_1^{ref} remains only a candidate. Paths and state descriptions are schematic.

4.3 Learning Target Selection

We construct learning targets by pairing a *learner* with a *donor*. A learner is a failed rollout that needs to make further progress, while a donor is another rollout from the same task group that provides a learning segment. Both successful and failed donors can supply supervision if they demonstrate the progress needed by the learner.

For each learner τ_i^{f}, we seek a segment demonstrating how to advance from its current frontier to the next milestone. Let k_i denote the position of its frontier in $\mathcal{M}$. The segment starts at m_{k_i} and ends upon reaching m_{k_i+1}. These boundaries define the relative learning interval (m_{k_i}, m_{k_i+1}), specifying the progress that a donor should demonstrate for the learner.

To obtain a concrete segment for this interval, we search the other rollouts in $\mathcal{G}_x^{\text{mix}}$ for donors that reach m_{k_i} and m_{k_i+1} in order. For a donor τ_j, let t_{start} denote the position matching the starting milestone and t_{end} the first subsequent position matching the target milestone. Its responses between these positions form the candidate segment provided by donor j for learner i,

$$\sigma_{j,i} = \left(u^j_{t_{\text{start}}}, \ldots, u^j_{t_{\text{end}}-1}\right). \tag{5}$$

Different donors may provide different candidate segments for the same interval, so we first check their compatibility with the learner. We require matching states at the starting milestone and that the first action in the segment be available in the learner observation. Let $\mathcal{V}_i$ denote the set of candidates satisfying these conditions. We then select the segment with the fewest responses,

$$\sigma_i^\star = \arg\min_{\sigma \in \mathcal{V}_i} |\sigma|, \tag{6}$$

where $|\sigma|$ is the number of responses in a segment. A failed donor can therefore provide a more suitable demonstration through a more direct local path. Learners with $\mathcal{V}_i = \emptyset$ are omitted.

4.4 POLICY UPDATE

To teach the model how to overcome its current bottleneck within its own experienced context, we concatenate the learner history with the selected donor segment to form a training sequence. For learner τ_i^{f}, $\mathrm{prefix}(\tau_i^{\mathrm{f}}, m_{k_i})$ denotes its history through the state matched to its relative frontier m_{k_i}. Appending the selected segment $\sigma_i^\star$ gives the training example

$$z_i = \mathrm{prefix}(\tau_i^{\mathrm{f}}, m_{k_i}) \parallel \sigma_i^\star, \tag{7}$$

where $\parallel$ denotes concatenation. The prefix provides the context in which the learner encounters the bottleneck, while the donor segment demonstrates how to reach the next milestone.

To focus training on overcoming this bottleneck, we compute the loss only on response tokens in the selected segment. The learner prefix and intervening environment observations serve only as context. We pool the examples across tasks into a training set $\mathcal{Z}$ and update the policy using masked cross-entropy. For token position p, let $z_{i,p}$ denote the current token and $z_{i,<p}$ its preceding context. We set $w_{i,p} = 1$ for response tokens in the selected segment and $w_{i,p} = 0$ otherwise. With θ denoting policy parameters and n_{sup} the total number of supervised tokens, the objective is

$$\mathcal{L}_{\mathrm{seg}}(\theta; \mathcal{Z}) = -\frac{1}{n_{\mathrm{sup}}} \sum_{z_i \in \mathcal{Z}} \sum_{p} w_{i,p} \log \pi_\theta(z_{i,p} \mid z_{i,<p}). \tag{8}$$

At evolution round r, we update π_r to π_{r+1} by optimizing the above loss on $\mathcal{Z}_r$, the training set constructed for that round. We then collect fresh rollouts with π_{r+1} and reconstruct the reference trajectories, relative frontiers, learning intervals, and donor segments for the next round. Repeating this process adapts supervision to the evolving policy.

5 EXPERIMENTS

We evaluate whether relative learning intervals provide effective supervision for language-agent self-evolution. Our experiments address three questions: **(1)** Does the proposed method improve task performance compared with learning from complete successful trajectories? **(2)** How far should supervision extend from a learner-relative starting point? **(3)** Does learning from failed-rollout segments outperform using successful-rollout segments alone?

5.1 EXPERIMENTAL SETUP

Environments and evaluation. We evaluate on ALFWorld (Shridhar et al., 2021) and WebShop (Yao et al., 2022), which require sequential interaction to complete tasks. We report success rate on both benchmarks and mean terminal reward (score) on WebShop. WebShop evaluation uses a fixed set of 500 test tasks. Comparisons use a common evaluation protocol. Both benchmarks use ReAct prompts (Yao et al., 2023), with two fixed in-context examples for ALFWorld and one for WebShop.

Model and training. We use Qwen2.5-7B-Instruct as the initial policy and update it through QLoRA-based supervised fine-tuning. ALFWorld uses four rollouts per task and WebShop uses eight; the reported evolution curves cover three and four rounds, respectively. Our method retains the learner prefix as context and supervises the selected donor interval. Training settings, loss masking, and weighting are described in Appendix A.

Supervision setting. Offline data construction uses rollouts generated by the policy, recorded environment states, and task-native terminal feedback. Recorded states support offline construction rather than serving as additional policy inputs. Our setting excludes human step annotations, stronger-teacher trajectories, and an external learned evaluator during task-level evolution.

Baselines. The comparison includes the initial policy, two successful-trajectory SFT baselines, and three literature baselines. Mixed-task SFT supervises the complete shortest successful trajectory from each mixed-outcome group $\mathcal{G}_x^{\mathrm{mix}}$. All-task SFT additionally includes the shortest successful trajectory from each all-success group $\mathcal{G}_x^{\mathrm{s}}$. Agent-R (Yuan et al., 2025) uses budget-constrained search to collect successful paths on ALFWorld and successful and revised trajectories on WebShop.

Table 1: Main results on ALFWorld and WebShop. ALFWorld reports success rates (%) for six task types and overall. WebShop reports terminal score on a 0–100 scale and success rate (%). Small subscripts show standard deviations. Bold means mark the highest mean in each column.

	ALFWorld							WebShop	
Method	Pick	Look	Clean	Heat	Cool	Pick2	Overall	Score	Success
Qwen2.5-7B-Instruct									
Base model	$86.46_{\pm 3.99}$	$75.00_{\pm 9.62}$	$78.23_{\pm 5.51}$	$77.17_{\pm 4.16}$	$85.71_{\pm 0.00}$	$57.35_{\pm 13.04}$	$77.61_{\pm 2.72}$	$12.42_{\pm 0.95}$	$3.85_{\pm 0.66}$
All-task SFT	$91.67_{\pm 0.00}$	$95.83_{\pm 5.32}$	$93.55_{\pm 0.00}$	$93.48_{\pm 2.51}$	$91.67_{\pm 4.56}$	$86.76_{\pm 7.40}$	$92.35_{\pm 0.71}$	$80.42_{\pm 0.73}$	$67.15_{\pm 0.53}$
Mixed-task SFT	$89.58_{\pm 4.17}$	$87.50_{\pm 8.33}$	$88.71_{\pm 4.16}$	$95.65_{\pm 5.02}$	$95.24_{\pm 3.89}$	$91.18_{\pm 7.59}$	$91.23_{\pm 3.19}$	$61.44_{\pm 1.42}$	$53.35_{\pm 0.75}$
Agent-R	$91.67_{\pm 5.89}$	$\mathbf{97.22}_{\pm 3.93}$	$91.94_{\pm 2.28}$	$\mathbf{100.00}_{\pm 0.00}$	$97.62_{\pm 3.37}$	$85.29_{\pm 12.48}$	$94.03_{\pm 2.11}$	$71.34_{\pm 1.56}$	$58.70_{\pm 0.42}$
ETO	$58.33_{\pm 5.89}$	$50.00_{\pm 0.00}$	$46.77_{\pm 11.40}$	$52.17_{\pm 0.00}$	$50.00_{\pm 3.37}$	$44.12_{\pm 12.48}$	$50.37_{\pm 0.53}$	$66.03_{\pm 1.38}$	$52.10_{\pm 0.42}$
ReST$^{\mathrm{EM}}$	$97.92_{\pm 2.95}$	$\mathbf{97.22}_{\pm 3.93}$	$\mathbf{96.77}_{\pm 0.00}$	$97.83_{\pm 3.07}$	$95.24_{\pm 0.00}$	$94.12_{\pm 8.32}$	$96.64_{\pm 0.53}$	$58.61_{\pm 1.67}$	$51.30_{\pm 1.56}$
Ours	$\mathbf{100.00}_{\pm 0.00}$	$95.83_{\pm 5.32}$	$\mathbf{96.77}_{\pm 3.72}$	$97.83_{\pm 4.35}$	$\mathbf{98.81}_{\pm 2.38}$	$\mathbf{97.06}_{\pm 3.40}$	$\mathbf{97.76}_{\pm 1.61}$	$\mathbf{82.75}_{\pm 0.67}$	$\mathbf{71.00}_{\pm 0.52}$

ETO (Song et al., 2024) constructs preference pairs from successful and failed trajectories for DPO training. ReST$^{\mathrm{EM}}$ (Singh et al., 2024) filters successful rollouts generated by the current policy for fine-tuning. All methods use the same base model and collect on-policy rollouts each round. Training continues from the previous checkpoint, except for ReST$^{\mathrm{EM}}$, which restarts from the base.

5.2 Overall Performance

Overall comparison. Our method achieves the best overall performance on both benchmarks (Table 1), supporting the use of relative progress to construct local learning targets from self-generated experience. Differences among baselines may reflect task structure and supervision design. Strong action dependencies in ALFWorld make complete successful trajectories useful as coherent demonstrations, which may explain the strong performance of ReST$^{\mathrm{EM}}$ in this environment. Agent-R enriches training experience through search and trajectory revision, but richer experience does not by itself specify what the current policy needs to learn. Our method compares progress across trajectories to determine both supervision boundaries, retaining the connected actions needed to cross a frontier while limiting behavior beyond the immediate learning need. This design accommodates both sequential decisions in ALFWorld and more local corrections in WebShop.

Comparison across evolution rounds. Figure 4 shows that fresh experience alone does not guarantee continued improvement; how supervision distinguishes useful behavior from failure also matters. ReST$^{\mathrm{EM}}$ achieves strong early performance on ALFWorld with smaller subsequent gains, while Agent-R improves across rounds in both environments. ETO improves on WebShop but declines on ALFWorld. Its trajectory-level preference objective treats an unsuccessful trajectory as the dispreferred alternative even when it contains useful local behavior. Such supervision may not clearly distinguish decisions responsible for failure from preceding useful progress. This could make learning signals less precise in ALFWorld, where actions are strongly interdependent, offering one possible explanation for the observed trend. Our method instead extracts segments that demonstrate further progress from failed trajectories and reconstructs learning targets after each policy update. Its continued improvement across both datasets validates the importance of local supervision.

5.3 Ablation Studies

To examine supervision scope and source, we compare relative learning intervals with next-action supervision, full-continuation supervision, and a successful-donor-only variant. Figure 5 reports success rates and cumulative supervised-token counts; exact values and construction details appear in Appendices B.2 and B.1.

Next-action supervision. Inspired by the single-action correction in UI-Voyager (Lin et al., 2026), this variant retains the same learning starting point but supervises only the next Thought–Action

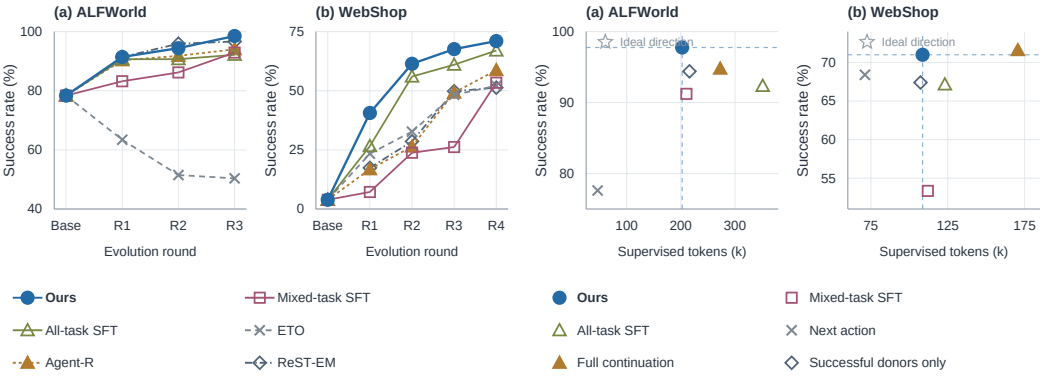


Figure 4: Comparison of success rate across evolution rounds on ALFWorld and WebShop. Panel axes use different ranges.

Figure 5: Success versus cumulative supervised tokens over three ALFWorld and four WebShop rounds. Panel axes use different ranges; the star indicates an ideal direction.

pair. Its effectiveness differs markedly across environments: next-action supervision approaches our performance on WebShop but performs substantially worse on ALFWorld.

To understand this contrast, we examine trajectory and segment lengths. Within mixed-outcome groups, the shortest successful trajectory per task averages 12.96 actions on ALFWorld R3, compared with 5.26 on WebShop R4. Our selected training segments average 8.26 and 2.74 actions, respectively (Appendix B.3). These differences suggest that a single correction may cover a larger share of the required local progress on WebShop, while providing insufficient guidance for the longer action sequences in ALFWorld. Relative learning intervals adapt the supervision scope to demonstrated progress instead of imposing a fixed single-action target.

Full-continuation supervision. Full continuation retains the failed rollout's prefix up to its frontier and supervises the remaining path to task completion using a segment from a successful rollout in the same group. Our method uses 25.79% fewer supervised tokens than full continuation on ALFWorld while increasing success from 94.78% to 97.76%. On WebShop, it uses 36.32% fewer tokens and reaches 71.00%, close to the 71.60% of full continuation. A complete continuation demonstrates a path to task success, but may also include behavior beyond the current local learning need. Relative intervals set an explicit local endpoint, concentrating training on the next increment of progress. These results suggest that effective supervision need not extend to task completion: an appropriate endpoint can yield better or similar performance with less supervision.

Successful donors only. Our method selects supervision segments from both successful and failed rollouts. To evaluate the benefit of using failed rollouts as donors, this ablation restricts segment sources to successful rollouts. Failed rollouts still define frontiers and provide learner context. Allowing failed donors improves success by 3.36 percentage points on ALFWorld and 3.60 on WebShop. A trajectory that fails overall can still contain useful progress that another attempt has not achieved. Restricting supervision to successful trajectories discards this experience, whereas selecting local segments by relative progress allows failed attempts to teach one another. The observed gains support this local value, which can arise from additional compatible segments and broader coverage of learning needs.

5.4 Behavioral Validity and Efficiency

To examine whether relative learning intervals improve execution efficiency alongside task success, we analyze interaction behavior on ALFWorld (Figure 6). Its longer action sequences make it particularly useful for assessing whether improved task completion is accompanied by more concise interaction. In these behavioral evaluation records, our policy reduces mean episode length by 10.21% and 8.44% relative to All-task SFT and Mixed-task SFT, respectively. Invalid-action rates also decrease by 85.47% and 86.89%, computed from the unrounded count ratios. Together, these results suggest that relative learning intervals support more efficient and reliable execution as well as improved task completion. Detailed results appear in Appendix E.

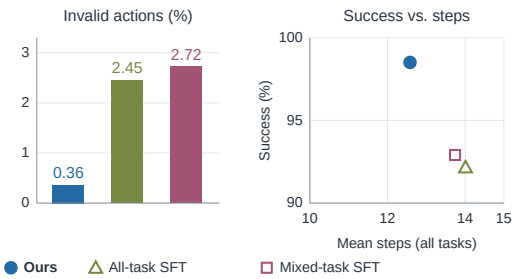


Figure 6: ALFWorld action validity and success versus mean steps. Step means include all episodes; upper left is better in the right panel.

Table 2: WebShop budgets: configuration = tasks/round × rollouts/task × rounds. Results use each configuration's final-round checkpoint. Score and success use a 0–100 scale.

Configuration	Rollouts	Score	Success
$250\times4\times4$	4,000	77.73	64.80
$250\times8\times4$	8,000	**82.75**	**71.00**
$500\times4\times2$	4,000	61.94	49.60
$500\times8\times2$	8,000	78.75	63.80

5.5 Effect of Rollout Budget

To examine how rollout budgets and their allocation across evolution rounds affect performance, we compare four configurations on WebShop. We sample either 250 tasks per round for four rounds or 500 tasks per round for two rounds, collecting four or eight rollouts per task in each setting. Table 2 reports final-round results for each configuration. Increasing rollouts per task from four to eight improves success by 6.20 and 14.20 percentage points, respectively. With equal total rollout budgets, the four-round configuration outperforms the two-round configuration in both score and success. These results suggest that self-evolution benefits depend not only on the amount of experience but also on the capability of the policy that generates it. When the initial policy has limited capability, more frequent updates using smaller batches of experience may build useful task capabilities earlier, enabling subsequent rollouts to provide more useful learning segments. Reconstructing policy-relative learning intervals from these fresh rollouts also keeps supervision aligned with what the updated policy needs to learn next. Together, these effects may reinforce the cycle of policy improvement and better training experience.

6 Discussion and Limitations

Our method adapts to different action dependencies by adjusting supervision scope through relative progress. In ALFWorld, an action often contributes to success only through a longer sequence of dependent decisions, so isolated corrections may provide insufficient guidance. Some WebShop attribute selections have more flexible ordering, making single-action supervision more effective. This contrast helps explain the differing performance of fixed supervision scopes and makes the two benchmarks informative tests of adaptability. Comparing progress across trajectories allows our method to select learning segments suited to these different requirements.

Our method remains limited by reference-dependent progress evaluation and residual path redundancy. First, tasks may admit multiple successful paths, but progress is measured against a single reference. Useful progress along alternative routes may therefore be underestimated, motivating evaluation that recognizes progress across paths. Second, relative comparison cannot remove unnecessary steps shared by the available trajectories. Combining segment selection with path optimization could further shorten learning targets while preserving task completion.

7 Conclusion

This paper identifies a central challenge in agent self-evolution: turning self-generated experience into supervision aligned with current learning needs. The proposed method uses relative progress across rollouts to determine both boundaries of a learning interval and extracts locally useful supervision from successful and failed trajectories. Experiments on ALFWorld and WebShop, together with ablations, support the effectiveness of this design in improving task performance and enabling iterative self-evolution. These results suggest that, alongside collecting more interaction experience, organizing available experience and specifying what to learn next are important factors in sustaining language model self-evolution.

AI USE STATEMENT

We used generative AI tools to assist with code development, polish the language of the manuscript, and produce preliminary figure drafts. All final figures were created by the authors. The core research ideas, conceptual framework, and original manuscript draft were developed and written by the authors. The authors take full responsibility for the final content of this work, including all AI-assisted code and text.

ETHICS STATEMENT

This work studies self-evolution methods for language model agents, with experiments conducted in the ALFWorld and WebShop benchmark environments. Given the scope and experimental setup of this study, we have not identified significant ethical risks that warrant specific discussion.

REPRODUCIBILITY STATEMENT

To support reproducibility, Section 4 describes the procedures for rollout collection, learning target construction, and policy updates. Section 5.1 specifies the experimental environments, base model, evaluation metrics, and number of rollouts per task. Appendix A reports training and evaluation settings, loss masking, and the use of fixed in-context examples. Appendix B.1 further describes the construction of the main ablation experiments. Code is available at `https://anonymous.4open.science/r/relative-frontier-evolve-657F/`.

REFERENCES

Sanjiban Choudhury. Process reward models for LLM agents: Practical framework and directions. *CoRR*, abs/2502.10325, 2025. doi: 10.48550/ARXIV.2502.10325. URL https://doi.org/10.48550/arXiv.2502.10325.

Ganqu Cui, Lifan Yuan, Zefan Wang, Hanbin Wang, Yuchen Zhang, Jiacheng Chen, Wendi Li, Bingxiang He, Yuchen Fan, Tianyu Yu, Qixin Xu, Weize Chen, Jiarui Yuan, Huayu Chen, Kaiyan Zhang, Xingtai Lv, Shuo Wang, Yuan Yao, Xu Han, Hao Peng, Yu Cheng, Zhiyuan Liu, Maosong Sun, Bowen Zhou, and Ning Ding. Process reinforcement through implicit rewards. *Trans. Mach. Learn. Res.*, 2026, 2026. URL https://openreview.net/forum?id=9SkkifLopZ.

Peng Cui and Mrinmaya Sachan. Investigating the zone of proximal development of language models for in-context learning. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Findings of the Association for Computational Linguistics: NAACL 2025, Albuquerque, New Mexico, USA, April 29 - May 4, 2025*, volume NAACL 2025 of *Findings of ACL*, pp. 6470–6483. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.FINDINGS-NAACL.362. URL https://doi.org/10.18653/v1/2025.findings-naacl.362.

Hanze Dong, Wei Xiong, Deepanshu Goyal, Yihan Zhang, Winnie Chow, Rui Pan, Shizhe Diao, Jipeng Zhang, Kashun Shum, and Tong Zhang. RAFT: reward ranked finetuning for generative foundation model alignment. *Trans. Mach. Learn. Res.*, 2023, 2023. URL https://openreview.net/forum?id=m7p5O7zblY.

Zi-Yi Dou, Cheng-Fu Yang, Xueqing Wu, Kai-Wei Chang, and Nanyun Peng. Re-rest: Reflection-reinforced self-training for language agents. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024*, pp. 15394–15411. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.EMNLP-MAIN.861. URL https://doi.org/10.18653/v1/2024.emnlp-main.861.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for LLM agent training. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5,*

2025, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/420c9f777c0b4f78d515e53cf74d58b2-Abstract-Conference.html.

Çaglar Gülçehre, Tom Le Paine, Srivatsan Srinivasan, Ksenia Konyushkova, Lotte Weerts, Abhishek Sharma, Aditya Siddhant, Alex Ahern, Miaosen Wang, Chenjie Gu, Wolfgang Macherey, Arnaud Doucet, Orhan Firat, and Nando de Freitas. Reinforced self-training (rest) for language modeling. *CoRR*, abs/2308.08998, 2023. doi: 10.48550/ARXIV.2308.08998. URL https://doi.org/10.48550/arXiv.2308.08998.

Amirhossein Kazemnejad, Milad Aghajohari, Eva Portelance, Alessandro Sordoni, Siva Reddy, Aaron C. Courville, and Nicolas Le Roux. Vineppo: Refining credit assignment in RL training of llms. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, volume 267 of *Proceedings of Machine Learning Research*. PMLR / OpenReview.net, 2025. URL https://proceedings.mlr.press/v267/kazemnejad25a.html.

Jiazheng Li, Yawei Wang, Qiaojing Yan, Yijun Tian, Zhichao Xu, Huan Song, Panpan Xu, and Lin Lee Cheong. SALT: step-level advantage assignment for long-horizon agents via trajectory graph. In Vera Demberg, Kentaro Inui, and Lluís Marquez (eds.), *Findings of the Association for Computational Linguistics: EACL 2026, Rabat, Morocco, March 24-29, 2026*, Findings of ACL, pp. 4709–4725. Association for Computational Linguistics, 2026. doi: 10.18653/V1/2026.FINDINGS-EACL.247. URL https://doi.org/10.18653/v1/2026.findings-eacl.247.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=v8L0pN6EOi.

Zichuan Lin, Feiyu Liu, Yijun Yang, Jiafei Lyu, Yiming Gao, Yicheng Liu, Zhicong Lu, Yangbin Yu, Mingyu Yang, Junyou Li, Deheng Ye, and Jie Jiang. Ui-voyager: A self-evolving GUI agent learning via failed experience. *CoRR*, abs/2603.24533, 2026. doi: 10.48550/ARXIV.2603.24533. URL https://doi.org/10.48550/arXiv.2603.24533.

Bo Qian, Yuting Wu, Shuang Zeng, Huaiyu Wan, Dalin Zhang, and Jiqiang Liu. Milegpo: Milestone inference with local evidence for graph-based policy optimization of long-horizon LLM agents. *CoRR*, abs/2608.19803, 2026. doi: 10.48550/ARXIV.2608.19803. URL https://doi.org/10.48550/arXiv.2608.19803.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *CoRR*, abs/2402.03300, 2024. doi: 10.48550/ARXIV.2402.03300. URL https://doi.org/10.48550/arXiv.2402.03300.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew J. Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=0IOX0YcCdTn.

Avi Singh, John D. Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Xavier Garcia, Peter J. Liu, James Harrison, Jaehoon Lee, Kelvin Xu, Aaron T. Parisi, Abhishek Kumar, Alexander A. Alemi, Alex Rizkowsky, Azade Nova, Ben Adlam, Bernd Bohnet, Gamaleldin Fathy Elsayed, Hanie Sedghi, Igor Mordatch, Isabelle Simpson, Izzeddin Gur, Jasper Snoek, Jeffrey Pennington, Jiri Hron, Kathleen Kenealy, Kevin Swersky, Kshiteej Mahajan, Laura Culp, Lechao Xiao, Maxwell L. Bileschi, Noah Constant, Roman Novak, Rosanne Liu, Tris Warkentin, Yundi Qian, Yamini Bansal, Ethan Dyer, Behnam Neyshabur, Jascha Sohl-Dickstein, and Noah Fiedel. Beyond human data: Scaling self-training for problem-solving with language models. *Trans. Mach. Learn. Res.*, 2024, 2024. URL https://openreview.net/forum?id=lNAyUngGFK.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. Trial and error: Exploration-based trajectory optimization of LLM agents. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pp. 7584–7600. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.ACL-LONG.409. URL https://doi.org/10.18653/v1/2024.acl-long.409.

Zhengwei Tao, Ting-En Lin, Xiancai Chen, Hangyu Li, Yuchuan Wu, Yongbin Li, Zhi Jin, Fei Huang, Dacheng Tao, and Jingren Zhou. A survey on self-evolution of large language models. *CoRR*, abs/2404.14387, 2024. doi: 10.48550/ARXIV.2404.14387. URL https://doi.org/10.48550/arXiv.2404.14387.

Gladys Tyen, Hassan Mansoor, Victor Carbune, Peter Chen, and Tony Mak. Llms cannot find reasoning errors, but can correct them given the error location. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, volume ACL 2024 of *Findings of ACL*, pp. 13894–13908. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.FINDINGS-ACL.826. URL https://doi.org/10.18653/v1/2024.findings-acl.826.

Hanlin Wang, Jian Wang, Chak Tou Leong, and Wenjie Li. Steca: Step-level trajectory calibration for LLM agent learning. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics, ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, volume ACL 2025 of *Findings of ACL*, pp. 11597–11614. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.FINDINGS-ACL.604. URL https://doi.org/10.18653/v1/2025.findings-acl.604.

Renxi Wang, Haonan Li, Xudong Han, Yixuan Zhang, and Timothy Baldwin. Learning from failure: Integrating negative examples when fine-tuning large language models as agents. *CoRR*, abs/2402.11651, 2024. doi: 10.48550/ARXIV.2402.11651. URL https://doi.org/10.48550/arXiv.2402.11651.

Zixuan Wang, Yuchen Yan, Hongxing Li, Teng Pan, Dingming Li, Ruiqing Zhang, Weiming Lu, Jun Xiao, Yueting Zhuang, and Yongliang Shen. Milestone-guided policy learning for long-horizon language agents. *CoRR*, abs/2605.06078, 2026. doi: 10.48550/ARXIV.2605.06078. URL https://doi.org/10.48550/arXiv.2605.06078.

Weimin Xiong, Yifan Song, Xiutian Zhao, Wenhao Wu, Xun Wang, Ke Wang, Cheng Li, Wei Peng, and Sujian Li. Watch every step! LLM agent learning via iterative step-level process refinement. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024*, pp. 1556–1572. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.EMNLP-MAIN.93. URL https://doi.org/10.18653/v1/2024.emnlp-main.93.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9,*

2022, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/82ad13ec01f9fe44c01cb91814fd7b8c-Abstract-Conference.html.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Siyu Yuan, Zehui Chen, Zhiheng Xi, Junjie Ye, Zhengyin Du, and Jiecao Chen. Agent-r: Training language model agents to reflect via iterative self-training. *CoRR*, abs/2501.11425, 2025. doi: 10.48550/ARXIV.2501.11425. URL https://doi.org/10.48550/arXiv.2501.11425.

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Chuanqi Tan, and Chang Zhou. Scaling relationship on learning mathematical reasoning with large language models. *CoRR*, abs/2308.01825, 2023. doi: 10.48550/ARXIV.2308.01825. URL https://doi.org/10.48550/arXiv.2308.01825.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. Star: Bootstrapping reasoning with reasoning. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/639a9a172c044fbb64175b5fad42e9a5-Abstract-Conference.html.

A Implementation and Evaluation Details

A.1 Training and Evaluation Settings

We use QLoRA-based supervised fine-tuning of Qwen2.5-7B-Instruct. Both Thought and Action tokens in selected donor segments contribute to the loss; learner-prefix, environment-observation, and other prompt tokens are masked. Core sampling, training, and evaluation settings appear in Table A.3. ALFWorld additionally uses within-segment position weighting (Appendix C); WebShop does not. WebShop assigns each retained task a normalized loss weight proportional to its failed-rollout count, divided equally among its supervised segments; ALFWorld uses no additional task-group weighting.

The seeds in Table A.3 are decoding seeds, not independent training replications.

Recorded ALFWorld states contain the complete set of simulator facts. WebShop uses a structured progress state containing the page role, product, selected options, and terminal purchase information. These states are used only for offline construction.

Fixed Prompt Examples. Both benchmarks use ReAct prompts (Yao et al., 2023): ALFWorld includes two fixed in-context examples, and WebShop includes one. These examples illustrate the interaction format and example task execution; they are prompt context and are excluded from the training loss.

ALFWorld Protocol. Each trajectory allows up to 50 environment actions and 100 model turns, with a 300-second timeout and at most 128 output tokens per model call. The interaction history is flattened into the prompt and left-truncated to retain the rightmost 4,096 input tokens.

WebShop Protocol. We use the official fixed 1,000-product catalog with synthetic goals. We generate a fixed list of 1,500 tasks, reserve the first 500 for testing, and use the remaining 1,000 as the training pool. Each round collects 250×8 rollouts from training tasks. Before each model call, we retain the most recent 13 history record blocks and then apply left truncation to keep the rightmost content within the 4,096-token input limit. Thus, the policy does not receive the full interaction history. Each trajectory is limited to 15 environment actions and a 600-second timeout; Thought generations do not count toward the action horizon. Thought and Action outputs are capped at 64 and 48 tokens per call, respectively.

Table A.3: Core sampling, training, and evaluation settings for the main experiments. Evaluation seeds refer to decoding replications of the trained checkpoint.

Setting	ALFWorld	WebShop
Training tasks per round	400	250
Rollouts per task	4	8
Evolution rounds	3	4
Effective batch size	8	8
Training epochs per round	1	1
Rollout temperature	0.7	1.0
Evaluation temperature	0.7	0.4
Evaluation tasks	134 (official unseen split)	500 (fixed test split)
Evaluation seeds	0, 1, 2, 3	0, 1, 2, 3

A.2 Baseline Implementation Details

We adapt the three literature baselines to the same base model and interactive tasks, preserving their main mechanisms for constructing learning signals and updating the policy. We run three evolution rounds on ALFWorld and four on WebShop.

Agent-R. We collect trajectories through budget-constrained Monte Carlo tree search. On ALFWorld, successful search paths provide the fine-tuning data. On WebShop, the current policy additionally evaluates and revises trajectories, and both revised and successful trajectories are used for training. Each round continues training from the previous policy, without mixing in additional general-purpose dialogue data.

ETO. We pair successful and failed trajectories for the same task as preferred and dispreferred examples and update the policy through DPO. The policy at the start of each round serves as the frozen reference model. After each update, fresh rollouts are collected to construct preference data for the next round.

ReST$^{\mathrm{EM}}$. We collect trajectories using the policy obtained in the previous round, filter successful trajectories according to terminal environment feedback, and use them for supervised fine-tuning. Although the sampling policy evolves across rounds, fine-tuning restarts from the base model each round rather than continuing from the previous trained parameters. This preserves the alternating process of sampling, filtering, and retraining.

B Mechanism Ablations and Supporting Analyses

B.1 Ablation Construction Details

The two mechanism ablations in Table B.4 cover ALFWorld and WebShop, with All-task SFT as the shared complete-trajectory baseline. Next action supervises one complete Thought–Action pair, not an Action-only target. This variant is inspired by the single-action corrective supervision in UI-Voyager (Lin et al., 2026), adapted to the learner contexts and interaction format used here.

The supervision-scope variants share the base model and first-round rollout pool, then collect their own rollouts and continue training from their respective previous-round checkpoints. They use the same frontier-construction procedure. Next-action and relative-interval supervision allow successful and failed donors. Full continuation reselects a compatible successful donor and follows it through terminal success. The variants need not share the same donor or a common intersection of learners. Prefix masking and optimization settings are held fixed.

In the successful-donor-only ablation, failed rollouts remain available for frontier construction and learner contexts. When a failed donor is excluded, the restricted variant selects a compatible successful donor using the same selection rules, if one is available. This restriction can affect both learner coverage and the selected local paths. The full-pool comparison measures their combined contribution; a comparison of donor quality on shared learners would require a separate matched subset.

B.2 Numerical Results for Supervision Scope and Donor Source

Table B.4 provides the numerical values plotted in Figure 5. Supervised-token counts are cumulative across three evolution rounds on ALFWorld and four on WebShop. Mixed-task SFT uses the shortest successful trajectory from each $\mathcal{G}_x^{\mathrm{mix}}$; All-task SFT additionally includes the shortest successful trajectory from each $\mathcal{G}_x^{\mathrm{s}}$. Success rates for Ours, Mixed-task SFT, and All-task SFT use the main-table means (Table 1); the ablation variants retain their reported results.

Relative to All-task SFT, our method uses 42.37% fewer supervised tokens on ALFWorld and 11.77% fewer on WebShop. A further comparison on ALFWorld holds the donor source to successful trajectories: successful donors only ends at the next milestone and uses 215,853 tokens, versus 272,662 for full continuation, a reduction of 20.83%. Their success rates differ by 0.38 percentage points. This numerical proximity suggests that local targets can retain much of the benefit of complete continuations, but does not establish statistical equivalence.

B.3 Successful Trajectory and Donor Segment Lengths

Table B.5 reports trajectory and segment lengths for ALFWorld R3 (400×4) and WebShop R4 (250×8). Panel (a) selects the shortest successful trajectory for each task with at least one successful

Table B.4: Success rates and cumulative supervised-token counts underlying Figure 5. Bold marks the highest success rate in each benchmark. Token counts measure supervision volume, not total computation.

	ALFWorld		WebShop	
Method	Success (%)	Tokens	Success (%)	Tokens
Ours	**97.76**	202,346	71.00	108,614
Mixed-task SFT	91.23	210,199	53.35	112,026
All-task SFT	92.35	351,101	67.15	123,097
Next action	77.61	46,247	68.40	71,125
Full continuation	94.78	272,662	**71.60**	170,571
Successful donors only	94.40	215,853	67.40	107,283

rollout and reports its mean action length, both over all such tasks and over mixed-outcome tasks alone. Panel (b) reports selected donor segment lengths, excluding the learner prefix, for all selected segments (including validation) and the training subset used for policy updates.

Table B.5: Successful trajectory and selected donor segment lengths. All lengths count environment actions. Panel (a) averages one shortest successful trajectory per task; counts refer to tasks. Panel (b) averages selected segments; counts refer to segments, and "all selected" includes training and validation splits. Statistics are specific to the indicated rounds, not cumulative across evolution rounds.

(a) Shortest successful trajectory per task

Benchmark / round	Tasks with success	Mean shortest length	Mixed tasks	Mixed mean shortest length
ALFWorld R3	369/400	9.30	106	12.96
WebShop R4	204/250	5.18	125	5.26

(b) Selected donor segments

	All selected (incl. validation)		Training only	
Benchmark / round	Mean length	Segments	Mean length	Segments
ALFWorld R3	8.37	193	8.26	174
WebShop R4	2.75	314	2.74	296

Both the shortest successful trajectories in mixed-outcome groups and the selected training segments are longer on ALFWorld than on WebShop in these rounds (12.96 versus 5.26 actions, and 8.26 versus 2.74 actions, respectively). These statistics provide descriptive context for the supervision-granularity ablation; the averages alone do not establish why a particular supervision scope performs better.

Interpreting supervision scope across environments. One possible explanation for the differing effects of next-action supervision is the dependency structure of task actions. After opening a WebShop product, some attribute choices can be made in different orders, so correcting one choice may provide useful local progress. In ALFWorld, obtaining and heating an apple requires a sequence of dependent actions; correcting an exploratory action alone may not demonstrate the transition needed for further progress. These examples motivate an interpretation rather than establish its cause. WebShop also uses eight rollouts per task versus four for ALFWorld, which can change the available reference frontiers. The observed segment lengths do not isolate task structure from sampling, and the scope comparisons can also differ in donor eligibility, retained coverage, and supervision volume.

C Effect of Within-Segment Position Weighting

Inspired by MileGPO (Qian et al., 2026), ALFWorld uses raw loss weights of 1.25 for the first Action and 1.5 for the final three Thought–Action pairs in each segment, taking the larger weight on overlap and normalizing by the mean weight over training target tokens. We remove this position weighting to test whether the gains over successful-trajectory SFT persist.

Table C.6 reports success rates from Base through R3. Without position weighting, our method reaches 95.90% at R3, exceeding Mixed-task SFT (92.91%) and All-task SFT (92.16%) by 2.99 and 3.74 percentage points, respectively. The weighted configuration reaches 98.51%, 2.61 points higher than the variant without position weighting. Thus, position weighting improves the reported final performance, but the advantage over these two SFT baselines remains when it is removed.

Table C.6: ALFWorld success rates (%) with and without within-segment position weighting. All configurations start from the same reported base success rate. The ablation removes positional loss weighting within learning segments.

Method	Base	R1	R2	R3
Ours	78.36	91.42	94.40	98.51
Ours w/o position weighting	78.36	91.42	92.54	95.90
Mixed-task SFT	78.36	83.21	86.19	92.91
All-task SFT	78.36	90.67	90.67	92.16

D Evolution Results and Budget Details

D.1 Performance Across Evolution Rounds

Figure 4 reports the base policy and each round through R3 on ALFWorld and R4 on WebShop, using the same evaluation tasks and protocol within each learning curve. Checkpoint initialization is described in Section 5.1 and Appendix A.2.

D.2 Supervision and Training Budgets

The cumulative supervised-token counts in Table B.4 count retained Thought and Action target tokens across evolution rounds. Prompt and observation tokens are excluded, and loss weights are not multiplied into token counts. These counts describe the retained supervision, not total training exposure after repeated epochs or inference-token usage.

D.3 Effect of Rollout Budget

The main-text budget comparison uses WebShop with four configurations: 250 tasks per round over four rounds or 500 tasks per round over two rounds, each with four or eight rollouts per task. Total collection budgets are 4,000 or 8,000 rollouts. Increasing rollouts per task can also increase retained supervision, so equal sampling protocols do not imply equal training exposure. Table 2 reports both metrics from the final-round checkpoint of each configuration.

E Action Validity and Completion Length on ALFWorld

Definitions and results. An action is counted as invalid if it is absent from the admissible-action list provided by the environment immediately before execution. The invalid-action rate is the total number of invalid actions divided by the total number of actions across evaluated episodes, rather than an average of per-episode rates. Table E.7 reports this rate alongside success and mean episode length over all tasks and over tasks completed successfully by each method. Our method has the lowest invalid-action rate (0.36%) and the shortest mean episode length over all tasks (12.58 steps).

Table E.7: Action validity and episode length on ALFWorld. Invalid-action counts are pooled over all evaluated actions. Successful-task means use episodes completed successfully by each method, not a shared-success subset.

Method	Success (%)	Invalid actions rate (count/total)	Mean steps all tasks	Mean steps own successes
Ours	**98.51**	**0.36% (12/3,372)**	**12.58**	12.09
All-task SFT	92.16	2.45% (92/3,756)	14.01	11.51
Mixed-task SFT	92.91	2.72% (100/3,683)	13.74	11.65

Progress toward success versus shortest-path behavior. Lower invalid-action rates indicate that the policy more often selects admissible actions; they do not establish that it follows shorter paths on every task. Computing mean length separately over the episodes completed successfully by each method gives 12.09 steps for our method, 11.51 for All-task SFT, and 11.65 for Mixed-task SFT. These averages condition on different sets of successful tasks and therefore do not isolate path efficiency on a common task set. Likewise, the all-task averages include both successful and unsuccessful episodes. A matched comparison on jointly successful tasks, with uncertainty estimates, would be needed to assess differences on that shared subset.

Our method teaches progress from the frontier of a learner to the next milestone while retaining its preceding context, which can include mistakes or detours. Selecting a short compatible donor segment does not minimize the length of the full execution. These results are consistent with improved progress toward success from imperfect histories, rather than uniformly shorter successful paths; this interpretation remains a hypothesis rather than a mechanism established by the aggregate metrics.