GO FOR THE REWARD: MONTE CARLO TREE SEARCH FOR TRAINING SUB-BILLION LANGUAGE MODEL AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Training sub-billion language model agents without external teachers is difficult when their initial policies rarely complete tasks and receive little guidance from task-level failure feedback. We introduce **Go for the Reward (GFR)**, a framework that learns directly from environment interaction without teacher demonstrations or task-specific supervised initialization. Inspired by the search–learning loop of Go-playing AI, GFR alternates training-time Monte Carlo tree search with policy learning. Search explores alternative continuations from shared interaction prefixes to discover successful trajectories. Execution constraints complement these trajectories with feedback on malformed or inadmissible actions and redundant interactions, including in unsuccessful attempts. The policy learns from this experience through successful-trajectory supervision, outcome-supported sibling-branch comparisons, and execution penalties, and then guides subsequent search. Starting from pretrained Qwen3-0.6B, GFR achieves 39.55% unseen success on ALFWorld and a WebShop Task Score of 59.61 without evaluation-time search. Both results are the highest among the compared teacher-free methods. On ALFWorld, this teacher-free training reaches performance comparable to teacher-assisted supervised fine-tuning followed by reinforcement learning. GFR also learns more efficient execution paths, reducing environment interactions by about 31% relative to its no-tree variant on a shared set of successfully completed WebShop tasks.

1 INTRODUCTION

Language model agents use tools, read environment feedback, and take sequences of actions to complete tasks (Yao et al. 2023b). Unlike single-response generation, these interactions require successive decisions that adapt to new observations. Repeated calls to large models are costly, motivating sub-billion language model agents specialized for particular tasks. Reinforcement learning (RL) can improve these agents through their own environment interactions (Feng et al. 2025). In multi-step environments with task-completion rewards, however, feedback reflects the outcome of an entire action sequence rather than individual decisions. When an initially weak policy rarely succeeds, repeated interactions provide mostly failure outcomes and few successful trajectories from which to learn.

For group-relative methods such as GRPO, a sampled group of failed trajectories provides no within-group advantage from task-completion rewards (Shao et al. 2024). Sampling more trajectories provides more chances to succeed, but a weak policy may repeat the same errors. Self-distillation uses the model itself to provide more detailed training signals. OPSD supplies reference solutions to the teaching policy, while SDPO supplies feedback or successful samples (Zhao et al. 2026; Hübotter et al. 2026). The model must still be able to interpret those solutions or feedback and produce useful teaching signals. SDPO's scaling experiments on coding tasks found larger gains over GRPO for larger models, but limited gains or worse results for smaller models (Hübotter et al. 2026).

External teachers can supply successful trajectories or token-level feedback. AgentTuning trains agents on filtered teacher-generated interactions, while GKD provides teacher feedback on student-generated sequences (Zeng et al. 2024; Agarwal et al. 2024). In our ALFWorld experiments, supervised training on successful teacher trajectories substantially improves Qwen3-0.6B's success

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

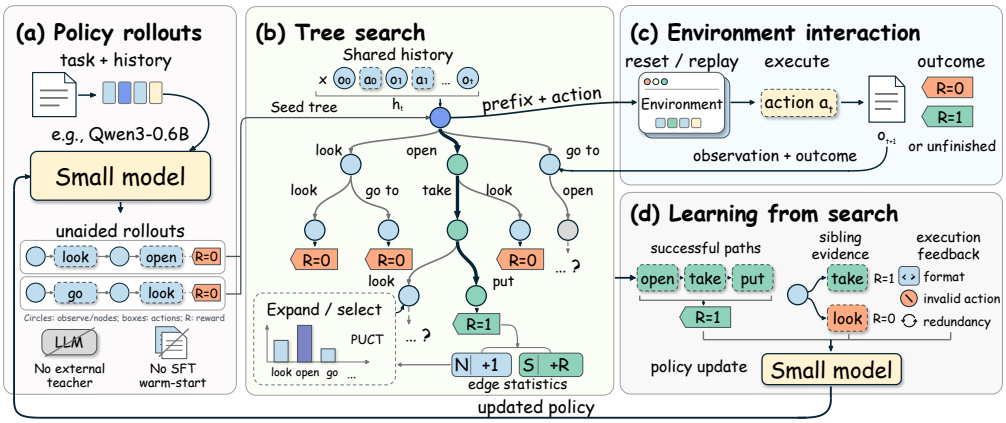


Figure 1: **Overview of Go for the Reward.** Unaided rollouts seed a shared search tree. Environment interaction supplies successful paths, sibling-branch outcomes, and execution feedback for policy updates; the updated model guides the next round of search. Circles represent decision nodes; action boxes show executed actions, with green highlighting successful paths.

rate. The model can then continue learning through RL on its own trajectories. These results suggest that the difficulty lies partly in obtaining useful experience, rather than only in the model's ability to learn from it. Teacher inference is still required to generate the demonstrations, but difficult to implement in many applications. We instead ask how a sub-billion language model can discover and learn from successful behavior through its own environment interactions.

Go-playing AI offers a model for combining search and learning. AlphaGo Zero and KataGo use the current policy to guide search, then train the network on search results (Silver et al. 2017; Wu 2019). Search can find behavior that the policy cannot yet reliably produce on its own. TS-LLM studies tree search for language-model generation and training, while Tree-GRPO derives policy-learning signals from agent trajectories with shared prefixes (Wan et al. 2024; Ji et al. 2026). We build on this search–learning loop to study how initially weak agents can acquire successful behavior and learn to execute it with fewer interactions.

We introduce **Go for the Reward (GFR)**, which combines Monte Carlo tree search (MCTS), learning from successful experience, and execution constraints (Figure 1). Search branches from shared action prefixes. Successful trajectories provide self-imitation targets, while sibling outcomes support penalties on alternatives. Execution constraints address format, admissibility, and redundancy, including in unsuccessful attempts. These signals update the policy that guides the next search round. GFR starts from a pretrained model, without an external teacher or a task-specific supervised fine-tuning (SFT) warm-start. The trained policy acts without search at evaluation.

Our main contributions are as follows.

1. We introduce GFR, a teacher-free framework that combines exploration, learning from successful experience, and execution constraints to bootstrap sub-billion language model agents.
2. We develop a training objective that reinforces verified successes, applies branch penalties when sibling comparisons support them, and learns from execution errors even when no trajectory succeeds.
3. With pretrained Qwen3-0.6B, GFR is competitive with teacher-assisted SFT followed by GRPO on ALFWorld. Ablations link search and the learning objective to shorter successful WebShop executions, and unlikelihood training to reduced repetition.

2 Related Work

2.1 Tree Search for Decision-Making and Learning

MCTS uses outcome backup to guide subsequent exploration (Coulom 2007). UCT and progressive widening control which actions to revisit and when to expand the search (Kocsis & Szepesvári 2006;

Couëtoux et al. 2011). Expert Iteration trains policies on search decisions (Anthony et al. 2017). AlphaGo, AlphaGo Zero, and AlphaZero combine neural policies, value estimation, and search; KataGo further improves self-play and training efficiency (Silver et al. 2016; Silver et al. 2017; Silver et al. 2018; Wu 2019). Together, these approaches establish a cycle in which search improves training targets and learned policies guide subsequent search.

For language models, search can improve both inference and training. Tree of Thoughts explores reasoning alternatives, while LATS combines MCTS with reflection and environmental feedback to improve decisions without parameter updates (Yao et al. 2023a; Zhou et al. 2024). TS-LLM and AlphaMath extend policy–value-guided search to language reasoning and train models from search-generated experience (Wan et al. 2024; Chen et al. 2024). TreeRL uses branches at uncertain reasoning steps to derive process-level training signals (Hou et al. 2025). These works establish that tree search can supply supervision as well as select outputs.

For interactive agents, a central design choice is how search outcomes guide policy updates. Agent Q uses guided MCTS and self-critique to construct off-policy DPO updates (Putta et al. 2024). Tree-GRPO derives intra-tree and inter-tree relative advantages (Ji et al. 2026). SEEA-R1 uses MCTS-derived action values for group-relative policy updates and jointly refines the policy and a multimodal generative reward model (MGRM), with variants using environment ground-truth rewards or model-estimated rewards (Tian et al. 2025). GFR reinforces environment-verified successes, prioritizes successful-path targets over sibling contrasts, and suppresses alternatives only when supported by branch comparisons. Execution errors supply additional training signals even when no trajectory succeeds.

2.2 Learning from Agent Experience

Experience can guide policy updates through relative returns or direct imitation of successful behavior. GRPO estimates relative advantages from sampled returns, while GiGPO adds step-level comparisons at shared states to improve credit assignment in multi-step agents (Shao et al. 2024; Feng et al. 2025). Self-imitation complements reward-based updates by retaining and reinforcing an agent's own high-return experience (Oh et al. 2018). For language models, STaR and ReST^{EM} develop this generate–verify–train loop through correct-answer rationales and reward-filtered samples, respectively (Zelikman et al. 2022; Singh et al. 2024). STaR additionally regenerates rationales conditioned on the correct answer when initial attempts fail. GFR combines successful-path self-imitation with tree-based exploration and feedback on execution errors.

Feedback can modify an agent's execution context or supply targets for parameter updates. Reflexion stores verbal reflections for subsequent trials, while GEPA, SkillOpt, and EvoTool optimize prompts, skills, or executable policies around a frozen model (Shinn et al. 2023; Agrawal et al. 2026; Yang et al. 2026b; Yang et al. 2026a). Re-ReST connects reflection to parameter learning by using environmental feedback to refine low-quality samples before self-training (Dou et al. 2024). OPSD and SDPO construct token-level teaching distributions conditioned on reference solutions or feedback and distill them on the policy's own rollouts (Zhao et al. 2026; Hübotter et al. 2026). GFR instead learns successful action sequences verified through environment execution, without constructing a feedback-conditioned teaching distribution.

3 Problem Formulation

We consider a language agent interacting with an environment to complete a task x. At step t, the agent receives an observation o_t and generates a response y_t from its interaction history $h_t = (x, o_0, a_0, \ldots, a_{t-1}, o_t)$. The response specifies an action a_t, which the agent executes to obtain the next observation:

$$y_t \sim \pi_\theta(\cdot \mid C(h_t)), \qquad o_{t+1} \sim \mathcal{E}_x(\cdot \mid h_t, a_t), \tag{1}$$

Here, π_θ is the language-model policy with parameters θ, C constructs the model input from the interaction history, and $\mathcal{E}_x$ is the environment's transition distribution for task x. The episode ends upon environment termination or at the action limit H, producing a trajectory τ with an environment-verified binary task-completion reward $R_x(\tau) \in \{0, 1\}$.

Our goal is to improve the policy's success rate without search at evaluation time:

$$J(\theta) = \mathbb{E}_{x\sim\mathcal{D}_{\text{eval}}}\mathbb{E}_{\tau\sim\pi_\theta,\mathcal{E}_x}\left[R_x(\tau)\right]. \tag{2}$$

Here, $\mathcal{D}_{\text{eval}}$ is the evaluation-task distribution, so $J(\theta)$ is the policy's expected standalone success rate. During training, the agent can reset the environment and replay action prefixes to explore alternative continuations.

4 Go for the Reward

Each GFR round collects unaided (core) trajectories, expands a shared tree for each task, and updates the model from observed outcomes (Figure 1). **Task learning** reinforces successful paths and applies branch penalties when sibling comparisons support them; **execution constraints** address format errors, inadmissible actions, and redundancy (Table 1).

Table 1: **Training signals and active losses.** CE: cross-entropy; Neg.: clipped negative loss; UL: unlikelihood. Positive format supervision applies only when no successful-path, sibling-branch, or redundancy target takes precedence.

Objective	Evidence	Active loss			Trained tokens
		CE	Neg.	UL	
Task learning	Success	✓	–	–	Response
	Sibling contrast	–	✓	–	Action content
Execution	Valid format	✓	–	–	Structure
	Format error	–	✓	–	Violation
	Inadmissible action	–	✓	–	Action content
	Redundancy	–	–	✓	Action content

4.1 Environment-Grounded Tree Search

At training round k, each node v for task x represents a full action prefix and each edge a policy-proposed action. Core trajectories initialize the tree; search resets the environment and replays prefixes to explore alternative continuations. Identical observations at different prefixes remain distinct.

Progressive widening allows the number of branches to grow sublinearly with completed visits (Couëtoux et al. 2011). New policy proposals must satisfy the environment's public action interface. When no new branch is added, PUCT selects among eligible edges (Silver et al. 2017):

$$a^* = \arg\max_{a\in\mathcal{A}(v)}\left[\widehat{Q}_k(v,a) + c_{\text{puct}}P_v(a)\frac{\sqrt{1+\widetilde{N}(v)}}{1+\widetilde{N}(v,a)}\right]. \tag{3}$$

Here, $\mathcal{A}(v)$ contains currently eligible edge actions, and a^* is the selected action. The branch-value estimate $\widehat{Q}_k(v,a)$ favors promising continuations, while the exploration term uses proposal prior $P_v(a)$ and strength c_{puct}. The prior is the smoothed frequency of policy proposals; $\widetilde{N}(v)$ and $\widetilde{N}(v,a)$ count completed and pending visits to the node and edge, respectively.

Completed paths update their executed edges with the environment's binary task-completion reward. Let $N(v,a)$ count distinct completed suffixes and $S(v,a)$ count successful ones. A fixed auxiliary Q-head prediction $q_k^{\text{head}}(v,a)$ contributes one prior pseudocount:

$$\widehat{Q}_k(v,a) = \frac{S(v,a) + q_k^{\text{head}}(v,a)}{N(v,a)+1}. \tag{4}$$

Policy and Q-head parameters remain fixed during collection; each completed attempt updates the tree statistics.

4.2 Task Learning from Interaction Outcomes

Let $\mathcal{T}_k$ index the training targets for round k, each associated with a response y_i, its history h_i, and a signal-specific token mask and weight. Here, i indexes training targets, not interaction steps. The subset $\mathcal{T}_k^{\text{succ}} \subseteq \mathcal{T}_k$ indexes eligible targets on successful trajectories, including retained successes from earlier rounds of the same run. The subset $\mathcal{T}_k^{\text{branch}} \subseteq \mathcal{T}_k$ indexes eligible targets on branches with lower observed success rates than a sibling with observed success, after applying the success and redundancy priorities below. Branch comparisons require completed outcomes on both edges; if no sibling has observed success, they provide no such target.

Successful behavior is reinforced through masked cross-entropy:

$$\ell_{\text{CE}}(i) = -\sum_{j \in \text{Tokens}_i} \alpha_{ij} \log \pi_\theta(y_{ij} \mid C(h_i), y_{i,<j}).$$

Here, y_{ij} is the j-th token of response y_i, and $y_{i,<j}$ is its preceding token sequence. The set Tokens_i contains the selected sampled-token positions, with coefficients α_{ij} normalizing answer and reasoning tokens separately and downweighting reasoning. Invalid actions and actions observed to be redundant are excluded from positive targets.

We adapt the clipped surrogate objective of PPO (Schulman et al. 2017) to eligible search-selected branches:

$$\ell_{\text{neg}}(i) = \max\{\rho_i, 0.8\}.$$

Here, ρ_i is the current-to-reference probability ratio over selected tokens, and θ_k denotes the fixed policy parameters at the start of round k. Minimizing this loss reduces the selected response's probability until the ratio reaches the clipping threshold; Appendix A.2 gives its exact definition.

The task-learning objective combines these targets:

$$\ell_{\text{task}}(i) = \begin{cases} \lambda_i \, \ell_{\text{CE}}(i), & i \in \mathcal{T}_k^{\text{succ}}, \\ \lambda_i \, \ell_{\text{neg}}(i), & i \in \mathcal{T}_k^{\text{branch}}, \\ 0, & \text{otherwise}, \end{cases} \tag{5}$$

with $\mathcal{L}_{\text{task}} = \sum_{i \in \mathcal{T}_k} \ell_{\text{task}}(i)$. The nonnegative weights λ_i are fixed during optimization. They balance successful targets across tasks and scale branch penalties according to the observed success-rate gap and the number of completed continuations. Success and redundancy targets take precedence over sibling-branch contrasts. Successful-path CE provides a learning signal even from a single discovered success.

4.3 Execution Constraints and Unlikelihood

Format checks, action admissibility, and observed redundancy provide training signals even when no trajectory succeeds.

Within $\mathcal{T}_k$, let $\mathcal{T}_k^{\text{format}}$ index positive format targets, $\mathcal{T}_k^{\text{invalid}}$ index localized format violations or inadmissible actions, and $\mathcal{T}_k^{\text{repeat}}$ index observed redundant actions. Format targets use ℓ_{CE} on structural tokens; violations use ℓ_{neg} on the affected tokens. Redundancy is penalized through action-content unlikelihood (Welleck et al. 2020):

$$\ell_{\text{UL}}(i) = -\log\big(1 - g_\theta(i)\big).$$

Here, $g_\theta(i)$ is the geometric mean of the redundant action's token probabilities, excluding reasoning and structural tokens. Minimizing UL lowers confidence in that action; its token-level definition is given in Appendix A.6. The execution-constraint objective is

$$\ell_{\text{exec}}(i) = \begin{cases} \lambda_i \, \ell_{\text{CE}}(i), & i \in \mathcal{T}_k^{\text{format}}, \\ \lambda_i \, \ell_{\text{neg}}(i), & i \in \mathcal{T}_k^{\text{invalid}}, \\ \lambda_i \, \ell_{\text{UL}}(i), & i \in \mathcal{T}_k^{\text{repeat}}, \\ 0, & \text{otherwise}, \end{cases} \tag{6}$$

with $\mathcal{L}_{\text{exec}} = \sum_{i \in \mathcal{T}_k} \ell_{\text{exec}}(i)$. Here, λ_i weights the applicable execution signal, with separate per-task caps for each signal type. Redundancy means read-only queries that repeat unchanged information in ALFWorld, or actions that leave the observed public page unchanged in WebShop. Search may also block such branches and select another policy proposal; core trajectories and evaluation use no action blocking.

4.4 POLICY AND Q-HEAD UPDATES

The policy is optimized jointly for task learning and execution constraints:

$$\mathcal{L}_\pi = \mathcal{L}_{\text{task}} + \mathcal{L}_{\text{exec}} + \beta\, \mathcal{D}_{\text{KL}}(\pi_{\theta_k} \| \pi_\theta). \tag{7}$$

The coefficient β controls the strength of the forward Kullback–Leibler (KL) regularizer, which limits policy drift from the fixed round-start reference. The divergence is evaluated over the full vocabulary at token positions included in the loss.

An auxiliary Q head predicts branch success from detached context–action representations, following value-guided search (Silver et al. 2017; Wan et al. 2024):

$$\mathcal{L}_Q(\phi) = \sum_{e \in \text{Edges}_k} u_e \, \text{BCE}\big(q_\phi^{\text{head}}(e), \bar{r}_e\big)\,. \tag{8}$$

Here, Edges_k is the set of edges with completed outcomes in round k, each written as $e = (v, a)$, and $q_\phi^{\text{head}}(e)$ is the head's success prediction with parameters ϕ. The target $\bar{r}_e = S(v, a)/N(v, a)$ is the observed success fraction for edge $e = (v, a)$, u_e its normalized evidence weight, and BCE denotes binary cross-entropy. Its fixed collection-time prediction is denoted by q_k^{head} in Equation (4); stop-gradient prevents its loss from updating the policy backbone. Equation (7) updates all policy parameters for the next collection round. Search parameters, target weights, replay limits, and optimization safeguards are specified in Appendix A.

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

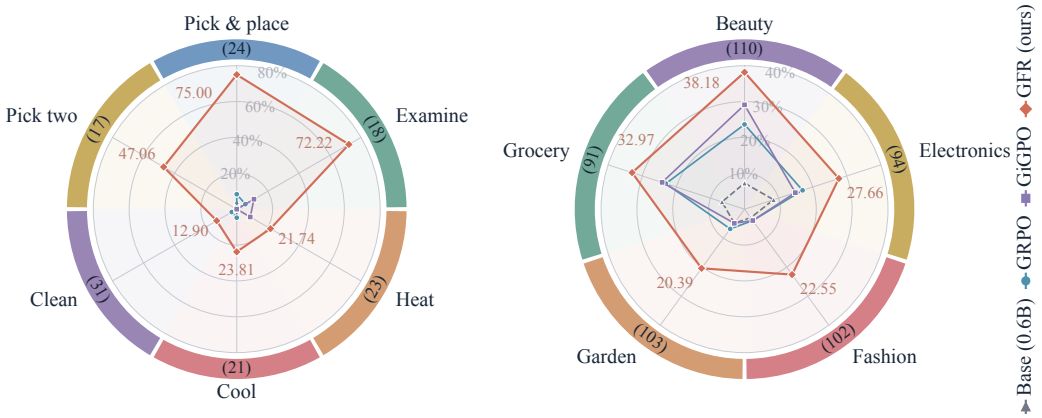
ALFWorld is a text-based household environment in which agents follow instructions to locate, manipulate, and place objects (Shridhar et al. 2021). WebShop is a simulated shopping environment in which agents search for products, inspect their attributes, and make purchases that satisfy user requests (Yao et al. 2022). GFR started from pretrained Qwen3-0.6B (Yang et al. 2025). We evaluated on the official ALFWorld seen and unseen splits and the WebShop test set. Evaluation used the trained policy without tree search, Q-head reranking, or search-time action blocking. We report task success rate, WebShop Task Score, and environment-action counts. Execution-length comparisons use tasks completed by every model in the comparison.

Table 2: **Standalone performance of Qwen3-0.6B agents.** Success requires full task completion; Task Score also measures partial completion (both 0–100). Actions is the mean over all 500 WebShop test tasks. Bold marks the best reported value per column; dashes denote unreported entries.

Method	Publication	ALFWorld		WebShop		
		Seen↑	Unseen↑	Success↑	Task Score↑	Actions↓
Base	—	1.43	0.75	6.00	17.02	13.27
Self-distillation						
SDPO	ICML '26	0.00	0.00	0.00	0.00	15.00
OPSD	ICML '26	0.00	0.00	0.00	0.00	15.00
Harness						
SkillOpt	arXiv '26	3.57	0.75	12.40	45.47	5.52
EvoTool	ACL '26	0.71	0.00	2.00	4.31	14.12
Reinforcement learning						
GRPO	arXiv '24	5.00	3.73	14.80	48.23	6.75
GiGPO	NeurIPS '25	2.86	2.99	15.40	49.97	7.09
T^2PO	ICML '26	0.71	0.00	15.60	48.13	7.00
Tree-GRPO	ICLR '26	6.43	3.73	**29.80**	56.14	8.53
GFR	Ours	**38.57**	**39.55**	28.40	**59.61**	**3.78**

Table 3: **Teacher-assisted references on ALFWorld.** Values are seen and unseen success percentages. SFT-success learns from successful teacher trajectories; + GRPO continues training that model through on-policy interaction.

Method	Publication	Teacher	Seen↑	Unseen↑
Qwen3-8B (teacher ref.; arXiv '25)		—	33.57	27.61
GKD	ICLR '24	Yes	27.86	0.00
GEPA	ICLR '26	Yes	2.14	2.24
SFT-success	—	Yes	30.71	37.31
+ GRPO	arXiv '24	Yes	37.86	37.31
GFR	Ours	No	**38.57**	**39.55**



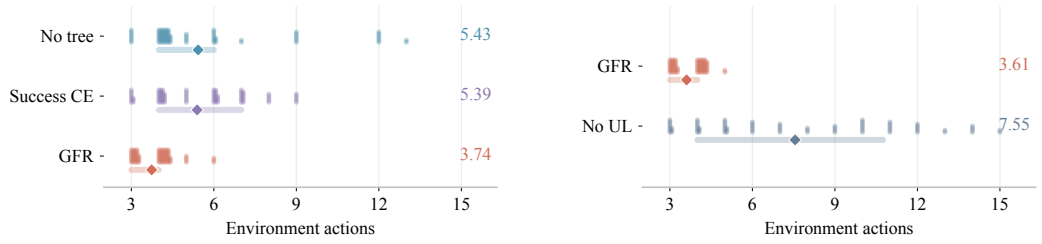
(a) ALFWorld (b) WebShop

Figure 2: **Success rates by category for Qwen3-0.6B agents.** Radial values show success percentages within each category. Outer sectors identify ALFWorld task types or WebShop product categories; parentheses give task counts, and point labels give GFR success percentages.

Comparisons cover group-based RL (GRPO, GiGPO), tree-based policy optimization (Tree-GRPO; Ji et al. 2026), turn-level exploration (T^2PO; Wang et al. 2026), self-distillation (SDPO, OPSD), and frozen-agent harness adaptation (SkillOpt, EvoTool; Yang et al. 2026b; Yang et al. 2026a). Teacher-assisted references are reported separately. Configurations, adaptation details, and budgets are in Appendices A and B.

5.2 Teacher-Free Bootstrapping

GFR enabled learning from an initial policy that rarely completed tasks (Table 2). On ALFWorld, unseen success increased from below 1% to 39.55%, while the compared group-based RL and

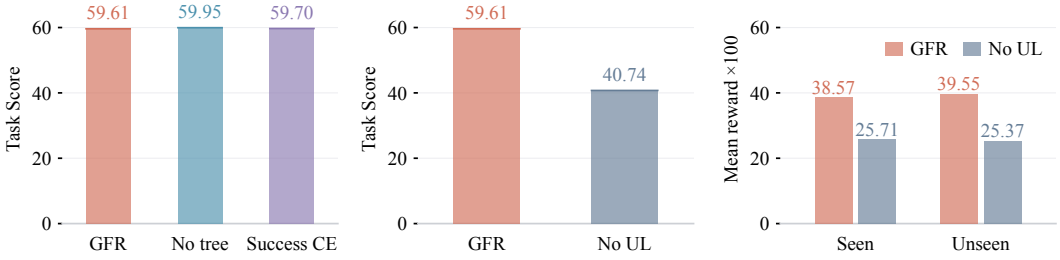


(a) Actions on 84 jointly solved WebShop tasks (b) UL ablation on 74 jointly solved tasks

Figure 3: **Execution length on jointly solved WebShop tasks.** Each panel includes only tasks completed by every method shown. Dots denote individual tasks, diamonds mark means, and bars show interquartile ranges; lower values indicate fewer environment interactions.

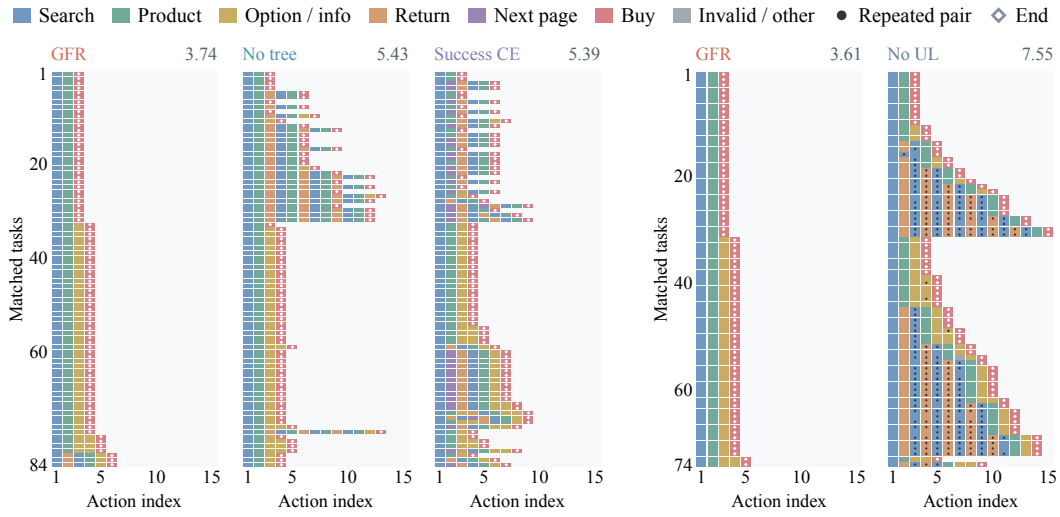
self-distillation baselines made limited progress from the same base model. These gains were retained without evaluation-time search. Tree-GRPO also struggled to learn from the weak initial policy in this setting. Its tree-based relative advantages still depend on reward differences: when all sampled trees fail with identical terminal rewards, they provide no task-advantage signal. GFR additionally learns from execution feedback in unsuccessful attempts. On WebShop, where the base model had a higher initial success rate, GFR improved both success rate and Task Score. The category-level comparisons in Figure 2 show that this improvement spans ALFWorld task types and WebShop product categories.

GFR achieved comparable ALFWorld performance to the SFT-success + GRPO reference without using teacher demonstrations (Table 3).



(a) WebShop: search/objective (b) WebShop: UL (c) ALFWorld: UL

Figure 4: **Task scores under training ablations.** Search/objective variants have similar WebShop scores; removing UL lowers both datasets' scores. ALFWorld's mean reward equals success rate.



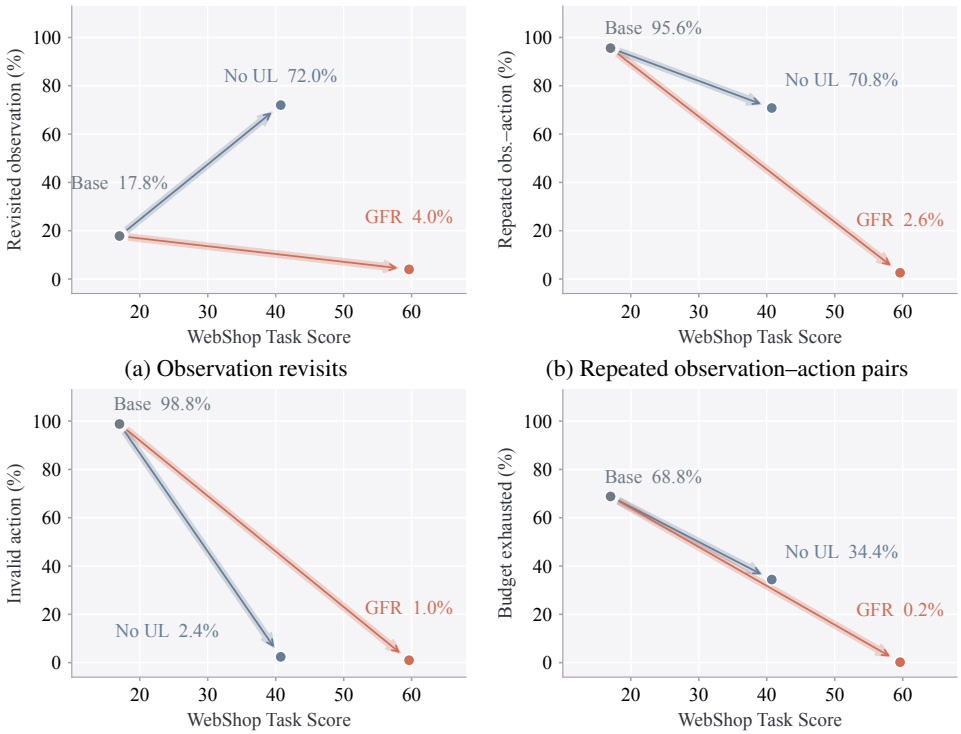
(a) Search and objective ablations: 84 shared successes (b) UL ablation: 74 shared successes

Figure 5: **Matched successful executions on WebShop.** Rows align the same tasks across methods within each group, ordered by GFR path length. Colored segments denote consecutive environment actions; dots in (b) mark repeated observation–action pairs. Headers give mean action counts.

5.3 Search and Learning Shape Execution

The WebShop ablations distinguish learning to complete a task from learning to complete it with fewer interactions. No tree replaces search with independent trajectories under the same nominal path budget; Success CE retains tree collection, successful-path CE, and format supervision, but removes branch, admissibility, and redundancy penalties (Appendix B.1). The resulting policies differ in execution length (Figure 3a), despite similar Task Scores (Figure 4a).

On tasks solved by all three policies, GFR uses about 31% fewer actions than No tree, while Success CE remains close to No tree (Figure 3a). The comparison with Success CE shows that the additional learning objectives help turn search experience into shorter executions. The execution paths show fewer repeated product searches and returns to earlier pages (Figure 5a).



(a) Observation revisits (b) Repeated observation–action pairs

(c) Invalid actions (d) Budget exhaustion without purchase

Figure 6: **Task Score and execution behavior on WebShop.** Arrows connect Base to independently trained GFR and No UL models. Vertical axes show task percentages. Revisits exclude unchanged observations; repeated pairs include them.

5.4 Learning to Avoid Redundant Actions

The No UL ablation retains search-time action blocking but produces much more repetitive standalone behavior (Figure 6). Repeated observation–action pairs occur on 70.80% of tasks without UL, compared with 2.60% for GFR.

No UL more often exhausts WebShop's action budget without purchasing and achieves a lower ALFWorld success rate and a lower WebShop Task Score (Figures 4b and 4c). On WebShop tasks completed by both policies, it uses more than twice as many actions as GFR (Figures 3b and 5b). UL therefore reduces redundancy in standalone execution, complementing action blocking during search.

6 Conclusion

We introduced Go for the Reward (GFR), a framework for training sub-billion language model agents without external teachers or a task-specific SFT warm-start. GFR combines training-time tree search with learning from successful trajectories and execution feedback. Execution constraints provide training signals even when task attempts fail, while search supplies successful continuations for policy learning. On ALFWorld and WebShop, the resulting policies improve over the pretrained model and operate without search at evaluation. GFR also achieves ALFWorld performance comparable to teacher-assisted SFT followed by GRPO. Beyond task completion, the WebShop comparisons show that GFR learns shorter execution paths, while unlikelihood training reduces redundant actions. These results suggest that combining search-generated experience with execution feedback can help weak initial policies learn to complete tasks and develop more efficient execution behavior.

AI Use Statement

Generative AI tools assisted with code development and language editing. The authors verified all AI-assisted content and take full responsibility for the final manuscript.

REFERENCES

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *International Conference on Learning Representations*, 2024.

Lakshya A. Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J. Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations*, 2026.

Thomas Anthony, Zheng Tian, and David Barber. Thinking fast and slow with deep learning and tree search. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Guoxin Chen, Minpeng Liao, Chengxi Li, and Kai Fan. AlphaMath almost zero: Process supervision without process. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-0870.

Adrien Couëtoux, Jean-Baptiste Hoock, Nataliya Sokolovska, Olivier Teytaud, and Nicolas Bonnard. Continuous upper confidence trees. In *Learning and Intelligent Optimization*, volume 6683 of *Lecture Notes in Computer Science*, pp. 433–445. Springer, 2011. doi: 10.1007/978-3-642-25566-3_32.

Rémi Coulom. Efficient selectivity and backup operators in Monte-Carlo tree search. In *Computers and Games*, volume 4630 of *Lecture Notes in Computer Science*, pp. 72–83. Springer, 2007. doi: 10.1007/978-3-540-75538-8_7.

Zi-Yi Dou, Cheng-Fu Yang, Xueqing Wu, Kai-Wei Chang, and Nanyun Peng. Re-ReST: Reflection-reinforced self-training for language agents. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 15394–15411. Association for Computational Linguistics, 2024. doi: 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 *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-1544.

Zhenyu Hou, Ziniu Hu, Yujiang Li, Rui Lu, Jie Tang, and Yuxiao Dong. TreeRL: LLM reinforcement learning with on-policy tree search. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12355–12369. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.604.

Jonas Hübotter, Frederike Lübeck, Lejs Behric, Anton Baumann, Marco Bagatella, Daniel Marta, Ido Hakimi, Idan Shenfeld, Thomas Kleine Buening, Carlos Guestrin, and Andreas Krause. Reinforcement learning via self-distillation. In *International Conference on Machine Learning*, 2026.

Yuxiang Ji, Ziyu Ma, Yong Wang, Guanhua Chen, Xiangxiang Chu, and Liaoni Wu. Tree search for LLM agent reinforcement learning. In *International Conference on Learning Representations*, 2026.

Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In *Machine Learning: ECML 2006*, volume 4212 of *Lecture Notes in Computer Science*, pp. 282–293, 2006. doi: 10.1007/11871842_29.

Junhyuk Oh, Yijie Guo, Satinder Singh, and Honglak Lee. Self-imitation learning. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 3878–3887. PMLR, 2018.

Pranav Putta, Edmund Mills, Naman Garg, Sumeet Motwani, Chelsea Finn, Divyansh Garg, and Rafael Rafailov. Agent Q: Advanced reasoning and learning for autonomous AI agents. arXiv preprint arXiv:2408.07199, 2024.

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

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

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

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529:484–489, 2016. doi: 10.1038/nature16961.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550:354–359, 2017. doi: 10.1038/nature24270.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419):1140–1144, 2018. doi: 10.1126/science.aar6404.

Avi Singh, John D. Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Xavier Garcia, Peter J. Liu, James Harrison, Jaehoon Lee, Kelvin Xu, Aaron Parisi, Abhishek Kumar, Alex Alemi, Alex Rizkowsky, Azade Nova, Ben Adlam, Bernd Bohnet, Gamaleldin 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. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856.

Wanxin Tian, Shijie Zhang, Kevin Zhang, Xiaowei Chi, Chun-Kai Fan, Junyu Lu, Yulin Luo, Qiang Zhou, Yiming Zhao, Ning Liu, Siyu Lin, Zhiyuan Qin, Xiaozhu Ju, Shanghang Zhang, and Jian Tang. SEEA-R1: Tree-structured reinforcement fine-tuning for self-evolving embodied agents. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-2628.

Ziyu Wan, Xidong Feng, Muning Wen, Stephen Marcus Mcaleer, Ying Wen, Weinan Zhang, and Jun Wang. AlphaZero-like tree-search can guide large language model decoding and training. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 49890–49920. PMLR, 2024.

Haixin Wang, Hejie Cui, Chenwei Zhang, Xin Liu, Shuowei Jin, Shijie Geng, Xinyang Zhang, Nasser Zalmout, Zhenyu Shi, and Yizhou Sun. T^2PO: Uncertainty-guided exploration control for stable multi-turn agentic reinforcement learning. In *International Conference on Machine Learning*, 2026.

Sean Welleck, Ilia Kulikov, Stephen Roller, Emily Dinan, Kyunghyun Cho, and Jason Weston. Neural text generation with unlikelihood training. In *International Conference on Learning Representations*, 2020.

David J. Wu. Accelerating self-play learning in Go. arXiv preprint arXiv:1902.10565, 2019.

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, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. arXiv preprint arXiv:2505.09388, 2025.

Shuo Yang, Caren Han, Xueqi Ma, Yan Li, Mohammad Reza Ghasemi Madani, and Eduard Hovy. EVOTOOL: Self-evolving tool-use policy optimization in LLM agents via blame-aware mutation and diversity-aware selection. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 43553–43572. Association for Computational Linguistics, 2026a. doi: 10.18653/v1/2026.acl-long.2016.

Yifan Yang, Ziyang Gong, Weiquan Huang, Qihao Yang, Ziwei Zhou, Zisu Huang, Yan Li, Xuemei Gao, Qi Dai, Bei Liu, Kai Qiu, Yuqing Yang, Dongdong Chen, Xue Yang, and Chong Luo. SkillOpt: Executive strategy for self-evolving agent skills. arXiv preprint arXiv:2605.23904, 2026b.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. WebShop: Towards scalable real-world web interaction with grounded language agents. In *Advances in Neural Information Processing Systems*, 2022.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, 2023a.

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

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. STaR: Bootstrapping reasoning with reasoning. In *Advances in Neural Information Processing Systems*, 2022.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. AgentTuning: Enabling generalized agent abilities for LLMs. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 3053–3077. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.findings-acl.181.

Siyan Zhao, Zhihui Xie, Mengchen Liu, Jing Huang, Guan Pang, Feiyu Chen, and Aditya Grover. Self-distilled reasoner: On-policy self-distillation for large language models. In *International Conference on Machine Learning*, 2026.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning, acting, and planning in language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 62138–62160. PMLR, 2024.

A Implementation Details

A.1 Main Configuration

Each run began with a fresh optimizer and empty success memory. The official splits contain 3,553 training, 140 seen, and 134 unseen ALFWorld tasks, and 10,587 training, 1,000 development, and 500 test WebShop instructions. Thinking was enabled under the token limits below. Evaluation used no tree search, Q-head reranking, or search-time action blocking. WebShop Task Score is the native fractional score multiplied by 100.

Table 4: **GFR training configuration.** Shared optimization and search settings, with environment-specific episode limits and outcome labels. Packing varies with available memory; the effective batch remains fixed.

Parameter	ALFWorld	WebShop
Initial policy	Qwen3-0.6B	Qwen3-0.6B
Teacher / SFT warm-start	None / none	None / none
Trainable parameters	Policy + Q head	Policy + Q head
Training seed	317701	317701
Rounds / tasks per round	75 / 16	75 / 16
Core / search attempts per task	2 / 14	2 / 14
Nominal path slots	19,200	19,200
Success label	Native success	Score = 1
Maximum actions	50	15
Response / thinking tokens	500 / 450	500 / 450
Temperature / top-p / top-k	1 / 1 / off	1 / 1 / off
Base policy learning rate	1e-6	1e-6
Effective optimizer batch	64 nodes	64 nodes
Accepted passes per round	1	1
AdamW betas	(0.9, 0.95)	(0.9, 0.95)
AdamW epsilon / weight decay	1e-8 / 0.01	1e-8 / 0.01
Gradient norm clip	1	1
KL coefficient	0.1	0.1
PUCT / prior smoothing	1.5 / 0.05	1.5 / 0.05
Widening coefficient / exponent	2 / 0.5	2 / 0.5
Evaluation-time search	None	None

ALFWorld interface and inherited defaults. We used the official ALFWorld splits and native environment success signal. Our interaction wrapper followed the base prompt and two-step history setting of verl-agent, the implementation accompanying GiGPO (Feng et al. 2025). Each decision included the task goal, current observation, admissible actions, and up to two preceding observation–action pairs. Earlier dialogue and generated reasoning were not accumulated as a full chat history. The 50-action episode limit also followed this configuration.

Prompt construction. Each decision is a single user message rendered with Qwen3's generation template. The initial ALFWorld observation contains the goal instruction; later prompts repeat that instruction and state the number of actions already executed. The prompt ends with the following instruction:

> 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, choose an admissible action and present it within `<action> </action>` tags.

WebShop prompts contain the shopping instruction, number of actions already executed, two recent observation–action pairs, current page, and available actions. Their output instruction is:

> Reason inside `<think> </think>` and choose exactly one action inside `<action> </action>`. Search queries must be your own text.

Thinking and response budgets. Qwen3's chat template used `enable_thinking=True`. Each response had a total limit of 500 generated tokens, including reasoning, delimiters, and action text. A decoding-time logits processor forced `</think>` at the 450-token thinking cap if reasoning had not already ended. The remaining response budget was available for the action and its delimiters; 450 tokens was an upper bound, not a required reasoning length.

Runtime packing and worker counts can vary with GPU memory while preserving the effective optimizer batch and target weights. The WebShop implementation uses the full native product catalog and the official instruction order. Its free-text search queries are proposed by the policy; public click targets determine click admissibility. Prompts and redundancy checks use the shopping instruction, current page, and interaction history. Generation uses vLLM; policy updates use PyTorch and

Hugging Face Transformers with scaled dot-product attention, bfloat16 autocasting, float32 trainable parameters, and gradient checkpointing. The Q head is trained in float32 on detached features.

A.2 Active Targets, Loss Operators, and Normalization

Each training target contains the original prompt, sampled response, a nonempty set of selected token positions Tokens_i, token coefficients α_{ij}, and nonnegative loss weight λ_i. Forced tokens are excluded. A target enters a loss only if recorded outcomes or execution feedback support it, its mask is valid, and its weight is nonzero. If no targets qualify, that loss term is zero.

The weights λ_i balance targets within each task; their construction and per-signal caps are specified in Appendices A.5 and A.6. The token coefficients α_{ij} determine positive-loss normalization.

We use the CE and clipped negative losses defined in Section 4.2. The probability ratio in the negative loss is

$$\rho_i = \exp(s_\theta(i) - s_{\theta_k}(i)), \qquad s_\theta(i) = \sum_{j \in \mathrm{Tokens}_i} \log \pi_\theta(y_{ij} \mid C(h_i), y_{i,<j}).$$

Here, $s_\theta(i)$ is the summed log probability over the selected tokens, and $\mathrm{clip}(z, l, u)$ bounds z to $[l, u]$. For a negative signed weight $A_i = -\lambda_i$, the identity $\lambda_i \ell_{\mathrm{neg}}(i) = -\min\{\rho_i A_i, \mathrm{clip}(\rho_i, 0.8, 1.2) A_i\}$ gives the implemented clipped surrogate loss. The log-ratio is numerically clamped to $[-20, 20]$ before exponentiation.

For success targets, α_{ij} implements the separate answer/reasoning normalization in Appendix A.5. For positive format targets it is one on selected structural tokens. The action-content UL operator is defined in Appendix A.6.

Loss selection avoids duplicate penalties. Successful responses receive successful-path CE; observed redundancy receives UL; otherwise, supported inferior branches receive the clipped negative loss. Format errors and inadmissible actions receive their respective penalties. Positive format targets apply only when no successful-path, sibling-branch, or redundancy target takes precedence. Fresh and replayed successful responses use the same CE loss but retain different per-task weights.

A.3 Update Acceptance

An optimizer window contains at most 32 steps, each accumulating an effective 64-node batch. KL probes include sampled reasoning positions, answer boundaries, and endings, not only the tokens selected for loss computation. A window is accepted when its mean probe KL is at most 0.02 and its maximum context KL is at most 0.1, relative to the round-start policy. Up to six window-level retries halve the policy learning rate; up to three whole-pass restarts reduce its starting learning rate by a factor of 0.25. Targets and their order remain fixed during a retry. The Q optimizer, policy optimizer, model parameters, random states, and accepted-step statistics are restored together.

A final generation check uses only contexts from the official training split. For n probe responses, the counts of admissible actions and responses with a nonempty action block must each remain at least half their pre-update count, rounded up to at least one. The count of responses with repeated action tags may not exceed the larger of $\lfloor 0.25n \rfloor$ and its pre-update value plus $\lfloor 0.15n \rfloor$.

A.4 Search Construction and Allocation

For a fixed task x, a node v is identified by its full executed action prefix $a_{<t}$. Its outgoing edges correspond to distinct policy-proposed actions at that prefix. Nodes with different prefixes remain distinct, including when their current observations match.

Core trajectories are inserted into the tree as executed. Subsequent attempts draw candidate actions from the policy and filter them using public admissibility information. Reusing an existing edge also preserves a stored policy response associated with that action and context.

Progressive widening. The open-ended text action space makes exhaustive expansion impractical. Let $N(v)$ denote the number of backed-up completed continuations through node v. The expansion limit is

$$W(v) = \left\lceil k_w \big(N(v)+1\big)^{\gamma_{\text{width}}} \right\rceil, \qquad k_w = 2, \quad \gamma_{\text{width}} = \tfrac{1}{2}. \tag{9}$$

Here, $W(v)$ bounds the number of branches, k_w sets its scale, and γ_{width} controls its growth with completed visits. In environments providing a finite admissible-action list, this limit is additionally capped by the list size. When the number of eligible edges is below the limit, a new admissible action becomes an edge and is executed for the first time. Otherwise, search selects an existing eligible edge. Actions blocked by the redundancy rule below are ineligible. Widening controls the number of retained policy-proposed alternatives.

PUCT allocation. Let $c(v,a)$ count how often the policy has proposed an existing edge action. We form a smoothed empirical proposal prior and renormalize it over eligible edges:

$$P_v(a) = \frac{c(v,a)+\epsilon}{\sum_{b\in\mathcal{A}(v)}[c(v,b)+\epsilon]}, \qquad \epsilon = 0.05. \tag{10}$$

Here, $\mathcal{A}(v)$ is the eligible-action set defined in Section 4.1, and ϵ is the proposal-smoothing constant. The branch-value estimate follows Equation (4), using completed-suffix counts and one fixed Q-head pseudocount.

When an attempt selects an existing edge, it maximizes

$$a^* = \arg\max_{a\in\mathcal{A}(v)} \left[\widehat{Q}_k(v,a) + c_{\text{puct}} P_v(a) \frac{\sqrt{N(v)+V(v)+1}}{1+N(v,a)+V(v,a)} \right], \qquad c_{\text{puct}} = 1.5. \tag{11}$$

Here, $V(v)$ and $V(v,a)$ count pending selections during parallel collection, so $\widetilde{N}(v) = N(v) + V(v)$ and $\widetilde{N}(v,a) = N(v,a) + V(v,a)$ in Equation (3). Pending visits affect edge selection, while completed outcomes determine success statistics. Pending counts are released when a search attempt finishes or fails. This allocation follows policy-guided tree search (Silver et al. 2017); the empirical proposal prior, single value pseudocount, and pending-visit handling above specify our implementation.

Outcome backup. A completed, verified path backs up its binary task-completion reward along its executed admissible edges. At each edge, an identical completed action suffix is counted only once. These statistics summarize the continuations selected by search. Attempts stopped with an unknown outcome count as neither success nor failure. Invalid responses can supply format evidence but do not create admissible tree edges. The influence of the prior q_k^{head} in Equation (4) diminishes as completed continuations accumulate.

Reused edges retain their original context and sampled tokens. Each completed attempt updates the tree and its N, S statistics under the suffix-deduplication rule above.

A.5 Task-Learning Targets and Loss Weights

This appendix details the targets and weights for task learning and execution constraints. Fresh and replayed successful responses form the success set; supported sibling-branch comparisons form the branch set.

Successful-path self-imitation. Let $\mathcal{V}_x^+$ contain nodes on verified successful trajectories for task x, after excluding invalid actions and actions with observed redundancy. Let $\mathcal{A}_v^+$ be the distinct successful actions observed at node v. One sampled policy response represents each action; additional reasoning variants or visits to the same edge do not increase its loss weight. Responses along a replayed prefix are included when their original prompts and sampled tokens are verified, so a successful continuation also trains earlier decisions.

For response y, let $\text{Tokens}_{\text{answer}}(y)$ contain sampled answer-token positions, including the action and its surrounding structure, and let $\text{Tokens}_{\text{reason}}(y)$ contain preceding reasoning-token positions. We omit the argument y in the following expression. Tokens inserted by the thinking-budget controller are excluded. We use

$$\begin{aligned}\ell_{\text{succ}}(\theta; v, y) = &-\frac{1}{|\text{Tokens}_{\text{answer}}|}\sum_{j\in\text{Tokens}_{\text{answer}}}\log\pi_\theta(y_j \mid C(h_v), y_{<j})\\ &-\frac{\eta}{|\text{Tokens}_{\text{reason}}|}\sum_{j\in\text{Tokens}_{\text{reason}}}\log\pi_\theta(y_j \mid C(h_v), y_{<j}), \qquad \eta = 0.1.\end{aligned} \tag{12}$$

Here, h_v is the interaction history at node v, y_j is the response's j-th token, and $y_{<j}$ is its preceding token sequence. This is ℓ_{CE} from Section 4.2 with separate answer and reasoning coefficients. The coefficient η downweights reasoning; an empty reasoning set contributes zero. Answer and reasoning lengths are normalized separately. For fresh evidence, each task with observed success contributes a total weight of one, divided equally over its successful nodes and then its distinct successful actions:

$$\mathcal{L}_{\text{succ}}^{\text{fresh}} = \sum_{x:\mathcal{V}_x^+\neq\varnothing}\frac{1}{|\mathcal{V}_x^+|}\sum_{v\in\mathcal{V}_x^+}\frac{1}{|\mathcal{A}_v^+|}\sum_{a\in\mathcal{A}_v^+}\ell_{\text{succ}}(\theta; v, y_{v,a}). \tag{13}$$

Here, $y_{v,a}$ is the retained successful response for action a at node v. The logarithms use full-vocabulary token probabilities. Thus, a node with only one successful action still supplies a nonzero learning signal.

Bounded same-run memory. Sparse successes should not disappear after one update. We retain a bounded memory of successful responses from earlier rounds of the same experiment, together with their exact prompts and task-prefix identities. Memory contributes a second term of the form in Equation (13), with per-task weight 0.25 rather than 1. Replay uses at most 25% as many loss-bearing tokens as the current round's fresh branch, execution, and success targets. The memory retains at most 512 nodes across 32 tasks, four actions per node, eight rounds of age, and three additional replay uses per response. Nodes already present in the fresh success set are not replayed again in that update. Redundant actions remain in the conditioning history but are excluded from positive targets. Replay suppression requires at least eight core trajectories for a task in the current round and at least 50% success; it is inactive with the main configuration's two core trajectories per task.

Supported branch contrasts. For each edge with completed outcomes, define its empirical success fraction $\bar{r}_{v,a} = S(v,a)/N(v,a)$. Among observed siblings, select a^+ with the highest fraction, breaking ties by completed-continuation count and action identity. A negative branch signal is available only when $\bar{r}_{v,a^+} > 0$, both edges have completed continuations, and the candidate edge has a lower success fraction. Before per-task normalization, its signed weight is

$$b_{v,a} = -0.1\,\frac{\min\{N(v,a), N(v,a^+)\}}{\min\{N(v,a), N(v,a^+)\}+2}\big(\bar{r}_{v,a^+} - \bar{r}_{v,a}\big). \tag{14}$$

Successful-path targets and verified redundancy evidence take precedence over this contrast. At the same node, actions with eligible successful-path targets in the current round receive equal CE weights rather than branch penalties, regardless of their empirical success fractions. Trees with no observed success provide no branch contrast. The count-dependent factor reduces the penalty when either branch has few completed continuations.

Per-task weight normalization. Weights for format, admissibility, sibling-branch contrast, and redundancy are normalized separately. If several response variants represent the same action at one prefix, their weight for each signal type is divided by the number of variants. For the resulting signed weight $d_{i,c}$ of target i and signal type c, let $\mathcal{T}_{k,x} \subseteq \mathcal{T}_k$ index targets belonging to task x. With per-task weight cap M_c, we use

$$\widetilde{d}_{i,c} = d_{i,c}\min\left\{1, \frac{M_c}{\sum_{r\in\mathcal{T}_{k,x}}|d_{r,c}| + \varepsilon}\right\}. \tag{15}$$

Here, r indexes targets in the same task; ε is a small positive constant that stabilizes the denominator, and $\widetilde{d}_{i,c}$ is the capped weight. Branch and admissibility each have cap 0.1 per task; redundancy has

its own 0.1 cap. Format uses the adaptive cap defined below. This prevents many response variants or repeated proposals for one task from giving that task an arbitrarily large loss weight. Successful responses are trained only through Equation (13); redundant actions are trained only through the UL loss.

For negative branch, format, and admissibility targets, use the signed weight $A_i = -\lambda_i < 0$ and probability ratio ρ_i defined in Appendix A.2. We minimize the clipped surrogate

$$\mathcal{L}_{\text{ctrl}} = - \sum_{i \in \mathcal{T}_k^{\text{branch}} \cup \mathcal{T}_k^{\text{invalid}}} \min \left\{ \rho_i A_i, \text{clip}(\rho_i, 0.8, 1.2) A_i \right\}. \tag{16}$$

This expression combines negative sibling-branch targets with negative format and admissibility targets. With nonnegative weight equal to the magnitude of each signed weight, it is exactly the weighted clipped negative loss defined in Appendix A.2.

Equation (16) adapts PPO clipping to a surrogate loss on responses collected during search, with token masks selected by the observed outcomes or execution feedback (Schulman et al. 2017). For branch targets, Tokens_i contains only sampled action-content tokens, excluding reasoning and action tags.

A.6 Execution Evidence and Token Masks

Format and redundancy targets use different token masks. The following definitions specify the execution-constraint losses.

Format supervision. Positive format supervision applies only to sampled action-tag tokens and the sampled final stop token. Negative format weights apply to an observed violation, such as a malformed tag or premature sampled stop. A response cut off by the length limit receives no format penalty unless a specific offending token can be identified. Correctly formatted but inadmissible actions receive a separate admissibility penalty, rather than a stronger format penalty. Structural masks exclude tokens whose decoded spans overlap action content. Before per-signal normalization, positive format targets receive weight $0.1(1 - f_k)$, where f_k is the task-balanced valid-format fraction among the round's recorded responses. Localized format violations receive $-0.1 \max(f_k, 0.1)$, and correctly formatted but inadmissible actions receive -0.1. Per-signal normalization precedes loss selection for successful responses and redundant actions.

The format cap adapts to the task-balanced format error e_k of the round's unaided core trajectories:

$$M_{\text{fmt},k} = 0.1 + 0.4 \text{ clip}\left(\frac{e_k - 0.05}{0.45}, 0, 1 \right). \tag{17}$$

Here, $M_{\text{fmt},k}$ is the per-task format-weight cap in round k. Unlike f_k, this cap uses only core trajectories and remains fixed during the update. Positive format targets use unclipped CE on their structural-token masks, weighted by the normalized positive format weight. Negative format targets use Equation (16). Structural-token supervision teaches the response format separately from action selection. Action tokens are sampled without decoding-time format constraints.

Detecting redundant actions. In ALFWorld, the detector tracks repeated outcomes of read-only `look`, `inventory`, and `examine` actions while the relevant observed information remains unchanged. Previous inventory results remain usable after movement; location-dependent query results are tied to the observed destination. Manipulation and unrecognized state-changing actions invalidate the stored query results that they may affect. In WebShop, the detector flags actions that leave the observed public page unchanged, excluding successful purchases.

A repeated matching outcome supplies local redundancy evidence. During search, an action with two matching prior query outcomes in its valid scope can be blocked; replacement must be an existing policy-proposed alternative at the same prefix. If no alternative exists, the attempt stops with an unknown outcome. Core trajectories and final evaluation do not use this intervention. Redundant actions are excluded from positive success targets, while the surrounding successful responses retain their original prompts.

Action-content unlikelihood. Motivated by unlikelihood training for repetitive generation (Welleck et al. 2020), we reduce the model's confidence in observed redundant actions. For each redundancy target i, $\mathrm{Tokens}_{\mathrm{action}}(i) = \mathrm{Tokens}_i$ contains only sampled action-content token positions. With normalized redundancy coefficient λ_i, the loss is

$$g_\theta(i) = \exp\left[\frac{1}{|\mathrm{Tokens}_{\mathrm{action}}(i)|}\sum_{j\in\mathrm{Tokens}_{\mathrm{action}}(i)}\log\pi_\theta(y_{ij}\mid C(h_i), y_{i,<j})\right],$$
$$\mathcal{L}_{\mathrm{UL}} = -\sum_{i\in\mathcal{T}_k^{\mathrm{repeat}}}\lambda_i\log(1-g_\theta(i)). \tag{18}$$

Here g_θ is the geometric mean of the action-content token probabilities. This adaptation avoids making the signal vanish simply because a response is long. Reasoning, action tags, and forced tokens are excluded. Numerical evaluation bounds the mean log probability below zero before applying the logarithm.

A.7 Q-Head Training and Policy Regularization

Architecture and inputs. The head is a two-layer perceptron, $d \to 128 \to 1$, with a SiLU hidden activation and sigmoid output, where d is the policy hidden size. Its input is the final non-padding hidden state of the decision prompt followed by `</think>` and `<action>`a`</action>`, without a generated reasoning trace. The output layer has zero initial weights and bias $\mathrm{logit}(0.05)$. Before a trained head is available, search uses a zero value prior. Thereafter, its parameters remain fixed during collection and are updated between rounds.

Training and gradient separation. The head in Equation (8) targets the observed success fraction among completed suffixes $\bar{r}_{v,a}$, with evidence weight $N(v,a)/(N(v,a)+2)$. We train it using binary cross-entropy, averaging eligible edges within a node and nodes within its optimizer batch. Edges without completed continuations have zero weight. A separate optimizer fits the head on detached backbone features, using AdamW with learning rate 10^{-6}, betas $(0.9, 0.95)$, epsilon 10^{-8}, weight decay 0.01, and gradient norm clipping at 1. Q-loss gradients do not update the policy backbone. Its predictions supply the value pseudocount in Equation (4), while observed outcomes determine policy targets.

The reference is the round-start policy. At scored token positions, the KL regularizer uses the full-vocabulary forward divergence

$$\mathcal{D}_{\mathrm{KL}}\left(\pi_{\theta_k}(\cdot\mid C(h_i), y_{i,<j})\,\|\,\pi_\theta(\cdot\mid C(h_i), y_{i,<j})\right). \tag{19}$$

Divergences are averaged over each training response's scored positions, using its token weights when applicable, and then over training targets in the optimizer batch.

The policy optimizer settings appear in Table 4; the acceptance and rollback procedure is specified in Appendix A.3.

B Additional Experimental Details and Results

B.1 Ablation Results

All training ablations started from pretrained Qwen3-0.6B and used 75 rounds with seed 317701. No tree replaced search attempts with independent trajectories under the same nominal path budget. Success CE retained tree collection, successful-path CE, format supervision, and detached Q-head learning, but removed sibling-branch penalties, admissibility penalties, and UL. It is a reduced-objective comparison, not an isolated sibling-branch ablation. No UL removed only the UL loss, retaining search-time action blocking and the other objectives. WebShop ablations use the binary-success main model as their comparator.

Table 5: **Search and task-learning ablations on WebShop.** Actions are averaged over the full test set, including unsuccessful episodes. Figure 3 compares execution lengths on jointly solved tasks.

Variant	Task Score↑	Actions↓
GFR	59.61	**3.78**
No tree	**59.95**	5.63
Success CE	59.70	6.35

Table 6: **UL ablation across datasets.** ALFWorld columns report success percentages; WebShop reports Task Score and mean actions over the full test set. No UL retains tree collection and search-time action blocking.

Variant	ALFWorld		WebShop	
	Seen↑	Unseen↑	Task Score↑	Actions ↓
GFR	**38.57**	**39.55**	**59.61**	**3.78**
No UL	25.71	25.37	40.74	9.87

B.2 Training Budgets and Comparison Scope

Each GFR run used 75 rounds of 16 training tasks, with two core trajectories and fourteen additional search attempts per task. This yields 1,200 task visits and 19,200 nominal path slots. Tasks were drawn from a seeded training-set permutation before repetition; the run did not exhaust either training split. GRPO and GiGPO used 150 rounds of 128 trajectories, also totaling 19,200 nominal paths. This matches path-slot counts, not generated tokens, environment transitions, optimizer work, or FLOPs. Tree attempts can share prefixes, terminate early, and incur environment replay. Their actual interaction and token costs therefore differ from independent trajectories.

Our Tree-GRPO adaptation started from pretrained Qwen3-0.6B and used 75 rounds of 16 tasks, with two trees and eight leaf-path slots per tree, also totaling 19,200 nominal paths. Each tree used seven expansions in one expansion iteration, with uniform internal-node selection including the root. Separately normalized intra-tree and inter-tree advantages were summed for policy updates. The adaptation used a PyTorch/DDP trainer rather than the upstream VERL runtime, with response packing and trajectory-token weighting. It used binary environment success, two-step history, and the same 500-token response and 450-token thinking limits, without an external teacher or GFR's success CE, UL, or Q head. Final standalone evaluation used the official ALFWorld seen/unseen splits and WebShop test set.

All reported training ablations used 75 rounds and the same nominal path-slot budget as GFR. No tree replaced the fourteen search slots with fourteen independent trajectories per task. T^2PO used 50 rounds with additional turn retries, which are not equated with independent paths. Harness methods used method-specific optimization budgets rather than a common RL-round count. Teacher-assisted references additionally required teacher inference and, where applicable, supervised training. These methods provide performance references rather than equal-compute comparisons. The experiments use one fixed seed; differences are reported descriptively, without claims of statistical significance across training seeds.

Additional baseline settings. GRPO and GiGPO use full-parameter updates. T^2PO includes token-level exploration controls. SkillOpt and EvoTool use the frozen 0.6B model for both acting and textual optimization. EvoTool uses eight generations with batch parameter three. SFT-success uses successful 8B-generated trajectories from the official training split; its GRPO continuation runs for 25 rounds.

B.3 Evaluation Protocol and Supplementary Results

Behavioral measurements. WebShop Task Scores and mean action counts include all 500 test tasks, including failures. Execution-length comparisons instead use the tasks jointly solved by the models in each panel. Category analyses use ALFWorld `valid_unseen` and WebShop test. Observation revisits count transitions to previously observed text, excluding unchanged observations. Repeated observation–action pairs include consecutive repetitions and invalid actions. Both metrics normalize

whitespace in observation text; pair matching also normalizes whitespace and lowercases action text. Invalid actions are executor rejections, covering malformed and inadmissible actions. Budget exhaustion means reaching 15 actions without purchase. Observation revisits alone need not be errors.

ALFWorld evaluation. Table 2 and Table 3 report success rates over 140 seen and 134 unseen tasks, rounded to two decimal places. GFR's final evaluation used temperature 1.0, top-p 1.0, and disabled top-k filtering, with one trajectory per task and seed 317701.

The pretrained Qwen3-0.6B and Qwen3-8B ALFWorld evaluation configuration used sampling with temperature 0.6, top-p 0.95, and top-k 20, one trajectory per task, and seed 317701. It retained the two-step history, 50-action limit, and 500/450 response/thinking budgets described above, without a one-shot demonstration. This sampling configuration is separate from GFR's temperature/top-p/top-k settings in Appendix A.1.

WebShop evaluations. The pretrained Qwen3-0.6B base completed 30/500 official test tasks, with Task Score 17.02. SDPO was evaluated at round 70 and completed 0/500 tasks, with Task Score 0.

WebShop reward variant. We also trained GFR from pretrained Qwen3-0.6B using native fractional returns, with the same 75-round, 19,200-path budget. Fractional returns replace binary outcomes in tree backup and Q-head targets; successful-path CE still requires full success. Table 7 compares this variant with the binary-success objective used in the main results and ablations.

Table 7: **WebShop reward variant.** Binary and native fractional training outcomes, evaluated without search. Success and Task Score use a 0–100 scale; all other training settings are unchanged.

Training outcome	**Success↑**	**Task Score↑**
Binary success (main)	28.40	59.61
Native fractional score	30.00	60.10