TSR: TRAJECTORY-SEARCH ROLLOUTS FOR MULTI-TURN RL OF LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Advances in large language models (LLMs) are driving a shift toward using reinforcement learning (RL) to train agents from iterative, multi-turn interactions across tasks. However, multi-turn RL remains challenging as rewards are often sparse or delayed, and environments can be stochastic. In this regime, naive trajectory sampling can hinder exploitation and induce mode collapse. We propose **TSR** (**T**rajectory-**S**earch **R**ollouts), a training-time approach that repurposes test-time scaling ideas for improved per-turn rollout generation. TSR performs lightweight tree-style search to construct higher-quality trajectories by selecting promising actions and trajectory prefixes during rollout generation. This improves rollout quality while preserving stable policy optimization and remains compatible with standard policy-gradient optimizers by design. Across Sokoban, FrozenLake, and WebShop, TSR achieves success-rate gains of up to 15 percentage points and converges in fewer optimization steps, while trading additional training-time rollout compute for stronger policies that require no search at inference time. By moving search from test time to the rollout stage of training, TSR provides a modular mechanism for stronger multi-turn agent learning, complementary to existing frameworks and rejection-sampling-style selection methods.

1 INTRODUCTION

Large language models (LLMs) are increasingly trained as interactive agents via reinforcement learning (RL), where the model acts over multiple turns in an environment and learns from feedback obtained through interaction rather than from static corpora (Nakano et al., 2022; Zhou et al., 2024; 2025). Recent works have taken strides in developing diverse practical frameworks for multi-turn RL training of LLM agents (Wang et al., 2025; Luo et al., 2025; Xi et al., 2025a; Cao et al., 2025; Cui et al., 2025; Zhu et al., 2025a). Across these systems, the training loop typically alternates between (i) generating rollouts for a batch of prompts, i.e., trajectories of agent–environment interactions, and (ii) updating the policy via policy gradient methods (e.g., PPO (Schulman et al., 2017) and GRPO (Shao et al., 2024)), using trajectory-level returns or per-turn rewards obtained from the environment.

Despite these advances, multi-turn RL remains brittle as per-turn rewards can be sparse or delayed, environments may be stochastic, and interaction horizons are often long. Under these conditions, naive rollout sampling, i.e., independently sampling full-length trajectories from the current policy, can leave advantage estimates at the mercy of low-quality or unlucky rollouts. This can amplify variance, slow exploration, and precipitate mode collapse known as *Echo Trap* (Wang et al., 2025), where model performance abruptly declines during training due to spikes in gradients. Several works have tried addressing the brittleness in RL, e.g., via turn-level credit assignment (Wei et al., 2025) and rejection-sampling-style rollouts such as GFPO (Shrivastava et al., 2025), RAFT (Dong et al., 2023), and StarPO (Wang et al., 2025). However, balancing exploration and exploitation, particularly in multi-turn environments, remains challenging. We provide an overview of related works in App. A.

In this work, we ask the following fundamental question: *Given additional compute allocated specifically to rollout generation, how much can we improve the performance and stability of training if we restructure how trajectories are generated?* Our answer is to move trajectory search from inference to training. Here, we draw inspiration from test-time scaling methods for reasoning (Snell et al., 2024), which have shown that spending additional compute budget on search can reliably

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.

improve solution quality at inference time. Specifically, we ask whether a similar compute-for-quality trade-off is applicable to multi-turn RL by upgrading the *per-turn rollout generator* itself.

To this end, we introduce **TSR** (**T**rajectory-**S**earch **R**ollouts), a train-time framework that adapts test-time search ideas to construct higher-quality trajectories. For each prompt, TSR runs a tree-style search over action sequences using available rewards or state-based feedback, and then retains the highest-scoring trajectories to compute advantages. By optimizing the trajectory *per-turn*, TSR avoids running into unlucky rollouts due to indiscriminate full-length trajectory sampling. As a result, TSR changes only how rollouts are generated, making it optimizer-compatible by design. This positions TSR within the family of search-guided policy improvement methods and is closely related in spirit to Expert Iteration (Anthony et al., 2017) and rejection-sampling-style fine-tuning, i.e., search improves the data distribution, while the policy optimizer distills the improved behavior back into the model.

In our experiments, we instantiate TSR with best-of-N, beam, and lookahead search methods, treating the corresponding search budgets (e.g., N, beam width, lookahead depth) as train-time compute knobs. We evaluate TSR for Qwen2.5-0.5B and Qwen2.5-3B models (Qwen et al., 2025) across three representative environments: (i) Sokoban (Weber et al., 2018), a deterministic logic puzzle, (ii) FrozenLake (Brockman et al., 2016), a stochastic RL game, and (iii) WebShop (Yao et al., 2023a), a long-horizon, agentic web navigation benchmark. TSR yields gains of up to 15 percentage points and converges in fewer optimization steps, without requiring additional search at inference time.

In summary, our main contributions are:

- **Search-Guided Rollout Generation.** We introduce TSR, an effective rollout generation framework for multi-turn RL that uses tree-style search techniques inspired from test-time scaling to construct higher-quality trajectories by optimizing rollouts at the *per-turn level*.
- **Modular and Structured Design.** TSR changes only *how* rollouts are generated, making it compatible with standard policy gradient methods (e.g., PPO, GRPO). We further show that its gains come from structured search rather than simply spending more compute on naive sampling.
- **Empirical Validation.** We demonstrate that TSR improves performance on deterministic, stochastic, and long-horizon tasks, enabling better performance with fewer interaction turns at inference.

2 Background and Problem Setup

We define preliminaries for multi-turn RL training and discuss the importance of high-quality rollouts.

2.1 Multi-Turn RL as a Partially Observable Markov Decision Process (POMDP)

Our focus is on agentic tasks, specifically multi-turn interactive decision-making, such as games, embodied navigation, or web-based interactions. Similar to (Xi et al., 2025a; Luo et al., 2025; Zhou et al., 2024), we model such multi-turn decision-making problems as a *Partially Observable Markov Decision Process* (POMDP) defined by the tuple $\mathcal{M} = \langle \mathcal{U}, \mathcal{S}, \mathcal{A}, \mathcal{O}, P, \mathcal{R} \rangle$, where $\mathcal{U}$ denotes the task space, $\mathcal{S}$ the state space, $\mathcal{A}$ the action space, and $\mathcal{O}$ the observation space. $P : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ represents the state transition function and $\mathcal{R} : \mathcal{U} \times \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ the reward function.

Given a task $u \in \mathcal{U}$, the LLM with policy π_θ parameterized by θ generates a first action $a_0 \sim \pi_\theta(\cdot \mid u)$ and its state is transitioned according to $s_1 \sim P(\cdot \mid s_0, a_0)$. A corresponding scalar reward $r_0 = \mathcal{R}(u, s_0, a_0)$ is provided to evaluate the quality of the action, and the agent receives an observation $o_0 \in \mathcal{O}$ from the environment. At each time step t, given the history $\tau_{<t} = (a_0, o_0, r_0, \ldots, a_{t-1}, o_{t-1}, r_{t-1})$, the agent generates the next action via $a_t \sim \pi_\theta(\cdot \mid u, \tau_{<t})$. This process continues for a maximum turn-length (horizon) K, resulting in the trajectory $\tau = (a_0, o_0, r_0, \ldots, a_K, o_K, r_K)$ with corresponding cumulative reward $R(\tau) = \sum_{t=0}^{K} r_t$.

2.2 Policy Optimization for Multi-Turn RL

Policy gradient methods (Sutton et al., 1999) aim to maximize the expected cumulative reward, defined as the objective function $J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)]$, via gradient ascent, i.e., $\theta_{\text{new}} = \theta + \eta \nabla_\theta J(\theta)$, where η is the learning rate. In the multi-turn setting, the policy gradient is given by $\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{K-1} \nabla_\theta \log \pi_\theta(a_t \mid \tau_{<t})\, \hat{A}_t\right]$, where $\hat{A}_t$ is the advantage estimate of taking action a_t.

In general, directly computing policy gradients is difficult because the gradient estimator typically suffers from high variance. To mitigate this, modern policy optimizers, such as PPO and GRPO, constrain the update step size and employ variance-reduction baselines for more reliable credit assignment. We provide details on how PPO and GRPO estimate advantages for multi-turn RL in App. B.1.

In this work, we do not alter these algorithms but instead modify the *rollout generation*. Rather than passively sampling complete trajectories from the current policy π_θ, we optimize rollouts on a per-turn level via search methods to improve the quality of the rollout data used for policy optimization.

2.3 Quality, Diversity, and Stability

The effectiveness and stability of multi-turn RL depend critically on the properties of the *rollout distribution*, which directly controls the signal-to-noise ratio of advantage estimates, and which regions of the state-action space receive a learning signal (Schulman et al., 2015; 2017; Mroueh et al., 2025). In agentic environments, however, naive stochastic rollouts often suffer from several issues:

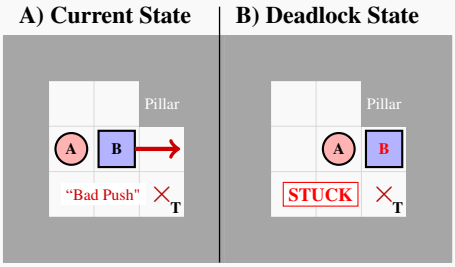
1. **Sparse or Delayed Rewards.** Per-turn rewards can be sparse, binary, or even delayed until the end of the episode. Consider a web agent that must purchase a specific item. Success requires a precise sequence of actions: *search for item, click on item page, click on right color, click on buy*. If the agent executes the first three steps perfectly but fails to click "Buy Now" at the final step, the reward at the trajectory level becomes zero. This makes advantage estimation challenging as Monte Carlo returns can have high variance and bootstrapped estimators depend on value predictions that are hard to learn when intermediate rewards are uninformative (Schulman et al., 2015; 2017).

2. **Irreversible Traps.** In many multi-turn environments, early mistakes can create irreversible states in which the episode becomes unsalvageable, collapsing learning (Ignatenko and Pravosud, 2023). To illustrate this, Fig. 1 shows how a single early mistake in the Sokoban game can render an episode unsalvageable, whereas simple best-of-N rollout selection recovers a feasible trajectory.

3. **Insufficient Diversity.** On-policy methods can become unstable if the collected rollouts lack diversity. For example, if the agent repeatedly samples the same successful trajectory for a given state, the relative advantage becomes zero ($\hat{A}_t \approx 0$), effectively stalling gradient updates. This lack of exploration leads to brittle policies that overfit to specific actions or shortcut behaviors rather than learning robust, generalizable heuristics. Thus, effective training requires a rollout distribution that balances *exploitation* (finding high rewards) with sufficient *exploration* (covering diverse logic paths) to prevent the policy from getting stuck in local optima.

Avoiding the "Corner Trap" in Sokoban

Task: Agent (A) must push box (B) into target (T). A "Pillar" blocks the path, requiring the agent to push the box downward first and then to the right to avoid a deadlock.



1. Naive Rollout. Agent prioritizes the immediate "valid" action: *Push Right*.

- **Mistake:** Box is pushed against the wall.
- **Deadlock:** To push the box down to ($\times$), the agent must stand on the cell above the box. However, that cell is occupied.
- **Outcome:** Box is frozen and the agent circles aimlessly until the episode ends.

2. Best-of-4 Rollout. Agent checks $N = 4$ different possibilities (up, down, right, left).

- **Exploration:** Some rollouts begin with *Push Right* and reach the same deadlock, while others first reposition the agent.
- **Selection:** Only rollouts that preserve access above the box can complete the task.
- **Outcome:** Best-of-4 selects a feasible trajectory and avoids the irreversible trap.

Figure 1: **"Corner Trap".** Naive rollout sees *Push Right* as progress but traps the box in a deadlock. Best-of-N explores multiple possibilities and selects one that avoids the dead-end.

These issues suggest that the *quality of rollouts* is a central factor in multi-turn RL. Motivated by the example in Fig. 1, we study whether allocating additional training-time compute to trajectory search can address these issues. Specifically, we formulate the following hypothesis:

Hypothesis: Search-Guided Rollout Optimization Improves Multi-Turn RL

In sparse-reward, multi-turn POMDPs, replacing naive stochastic rollout sampling with search-guided, per-turn trajectory optimization yields a rollout distribution with higher effective learning signal and improved stability, without modifying the policy optimizer.

We test this hypothesis by augmenting rollout generation via tree search guided by available step-level rewards or state-based feedback. These signals are used only for rollout construction, whereas policy optimization remains based on the original task reward. This repurposes test-time search for train-time data collection, analogous to recent process-reward-guided tree search methods for collecting higher-quality reasoning traces (Yao et al., 2023b; Zhang et al., 2024).

3 Optimizing Rollouts with Trajectory Search

In this section, we formalize rollout generation as a trajectory search problem and introduce TSR.

3.1 Trajectory Rollouts via Tree Search

We consider multi-turn rollout generation as a sequential decision problem that induces a search tree over action sequences (see Fig. 2). Starting from an initial state s_0, each node corresponds to a partial trajectory prefix $\tau_{<t}$, and edges correspond to candidate actions sampled from the current policy.

Candidate Action Set. At each turn t, instead of sampling a single action from the corresponding policy $\pi_\theta(\cdot \mid \tau_{<t})$, we sample a *candidate action set* $\mathcal{A}_t = \{a_t^{(1)}, \ldots, a_t^{(M)}\}$, $a_t^{(j)} \sim \pi_\theta(\cdot \mid \tau_{<t})$, where M controls the per-turn expansion budget. This corresponds to expanding a node in the tree.

Scoring Function. We evaluate the desirability of taking action a_t at prefix $\tau_{<t}$ with outcome o_t from the environment using a *scoring function* $S(\tau_{<t}, a_t, o_t) \in \mathbb{R}$. In agentic environments, $S(\cdot)$ scores the outcome of an executed action, e.g., the state, observation, or task progress, rather than arbitrary partial text prefixes. When available, $S(\cdot)$ may simply correspond to the step-level environment reward. When immediate rewards are sparse or delayed, it may instead incorporate heuristic signals, learned value estimates, or state-based feedback from the simulation. This abstraction allows our framework to support (1) *reward-based scoring* in deterministic settings (e.g., logic puzzles such as Sokoban), (2) *heuristic or risk-aware scoring* in stochastic environments (e.g., FrozenLake), and (3) *progress-based or semantic scoring* in delayed-reward settings (e.g., WebShop).

Trajectory Search Rollouts (TSR). Given the candidate action set and scoring function, rollout generation selects actions according to a *search strategy* $\mathcal{F}_\phi$ parameterized by search parameters ϕ:

$$\text{TSR} : (\pi_\theta, u, S, K, \mathcal{F}_\phi) \longrightarrow \{\tau_1, \ldots, \tau_L\}\,, \tag{1}$$

where $u \in \mathcal{U}$ denotes a task instance with initial state s_0 and L denotes the number of retained trajectories. By repeatedly expanding candidate actions and selecting according to the search strategy $\mathcal{F}_\phi$, which reweights the rollout distribution toward trajectories with higher-scoring prefixes, rollout generation corresponds to traversing a truncated tree. Thus, TSR induces a search-guided rollout distribution $\mu_{\theta,\phi}(\tau \mid u; S)$. Because all candidate actions are proposed by the current π_θ, moderate search can remain policy-proximal (see Sec. 5.5 and App. F.4). Crucially, TSR modifies only the *rollout generation process*, i.e., the policy optimizer, objective, and update rules remain unchanged.

3.2 Search Strategies

We instantiate TSR with best-of-N, beam, and shallow lookahead search. While these are popular test-time scaling techniques traditionally applied at inference time, here we adapt them to *training-time rollout generation* for multi-turn RL. Fig. 5 in App. C provides an overview of these adaptations.

Trajectory-Level Best-of-N. For a given initial state, we independently sample a set of N *complete* trajectories $\{\tau^{(1)}, \ldots, \tau^{(N)}\}$ from the current policy π_θ, where each trajectory is scored using its

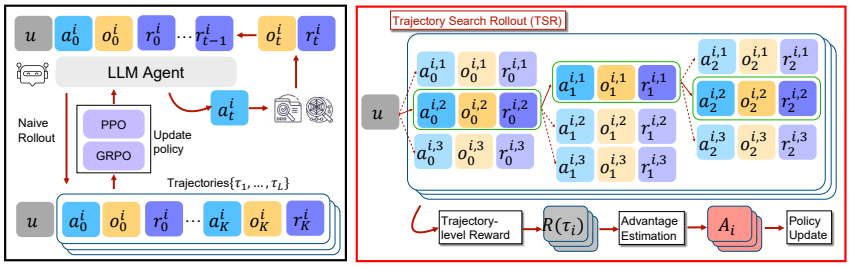


Figure 2: (**Left**) Multi-turn RL with naive rollouts: trajectories are sampled independently without any search. (**Right**) Trajectory Search Rollouts (TSR): using tree-style search to construct high-quality trajectories by selecting high-scoring actions at each turn.

cumulative reward $R(\tau)$. Rollout generation then retains the top-L trajectories for policy optimization:

$$\mathcal{T}^* = \mathrm{Top}_L\left(\{\tau^{(1)}, \ldots, \tau^{(N)}\}; R\right). \tag{2}$$

Because selection occurs only *after* full trajectories are generated, it does not influence intermediate action choices and may still generate suboptimal prefixes. We therefore view trajectory-level best-of-N as a flat search baseline with branching only at the root and selection after complete rollouts.

Per-Turn Beam Search. At each turn t, beam search maintains a set $\mathcal{B}_t$ of B active partial trajectories and allocates a total budget of M candidate actions across them. Let $\mathcal{A}_t(\tau)$ denote the candidates sampled for prefix τ, with $\sum_{\tau \in \mathcal{B}_t} |\mathcal{A}_t(\tau)| = M$. Each candidate is scored using $S(\tau, a_t, o_t)$, and the top-B resulting prefixes are retained:

$$\mathcal{B}_{t+1} = \mathrm{Top}_B\Big\{\tau \circ a_t \;\Big|\; \tau \in \mathcal{B}_t,\ a_t \in \mathcal{A}_t(\tau)\Big\}, \tag{3}$$

where $\circ$ denotes concatenation and ranking uses accumulated scores. After K turns, the B surviving trajectories are retained. To obtain L trajectories per task group, we run L/B independent beam searches. Thus, beam search uses approximately $(L/B)MK$ candidate transitions compared with LK for naive rollouts. For $M = 4$ and $B = 2$, this yields a relative rollout-generation budget of $M/B = 2\times$ over the standard instance-filtering baseline introduced in Sec. 3.3.

Shallow Lookahead Search. Similar to beam search, shallow lookahead evaluates candidate actions at each turn but additionally executes a short continuation of future steps (with depth $D \ll K$) before selection. This corresponds to evaluating a truncated subtree rooted at $\tau_{<t}$ and choosing actions based on their downstream outcomes. Because candidates and continuations are generated in a lightweight action-only format, its compute per update step remains comparable to beam search despite a larger transition count (see App. E).

3.3 Instance Filtering for Enhanced Task Diversity

While TSR improves the quality of trajectories (*exploitation*), it does not by itself guarantee sufficient *task-level* diversity (*exploration*). An established mechanism to promote such diversity is *instance-level filtering* (Wang et al., 2025), which hierarchically samples distinct task realizations before generating corresponding rollouts. Specifically, at each training iteration, instance filtering first samples P task instances (or *groups*), where each group corresponds to a different initial state (e.g., different Sokoban layouts or WebShop tasks). Within each group u with initial state s_0, the agent generates L corresponding rollout trajectories $\{\tau^{(1)}, \ldots, \tau^{(L)}\} \sim \pi_\theta(\cdot \mid u)$, resulting in a total of $P \times L$ trajectories per iteration. For each group, instance filtering then defines an *outcome uncertainty* under the current policy π_θ as the standard deviation of resulting trajectory returns, i.e.,

$$U(u; \pi_\theta) = \mathrm{Std}_{\tau \sim \pi_\theta(\cdot|u)}\big[R(\tau)\big]. \tag{4}$$

During training, groups are ranked according to $U(u; \pi_\theta)$, and only the top-$p\%$ most uncertain groups are retained for policy updates, while low-variance groups are discarded. The motivation for group-based instance filtering stems from active learning (Settles, 2011), where groups with low reward variance tend to be uninformative, i.e., trajectories either solve trivial problems (uniformly high reward) or fail consistently (uniformly low reward), providing little learning signal.

Empirically, Wang et al. (2025) show that uncertainty-based filtering can prevent mode collapse by maintaining a diverse training set. We therefore combine TSR with instance filtering to jointly optimize rollout quality and diversity. Alg. 1 summarizes the joint operation.

Since TSR is modular, it can be combined with other mechanisms that encourage exploration (e.g., ScalingInter-RL (Xi et al., 2025a)) or hierarchical RL methods (e.g., (Luo et al., 2025)). A systematic study of integrating TSR with such methods is an interesting future work.

Algorithm 1 TSR with Instance-Level Filtering

Input: Policy π_θ, Task distribution $\mathcal{D}$, Sample Size P, Trajectories L, Filter Ratio p, Strategy $\mathcal{F}_\phi$, Turn Horizon K, Score Function S.
Output: Optimized Policy π_{θ^*}
for each training step **do**
  Sample P task groups $\{u_1, \ldots, u_P\} \sim \mathcal{D}$.
  for each group u_i **do**
    // Generate L trajectories using TSR:
    $\mathcal{G}_i \leftarrow \text{TSR}(\pi_\theta, u_i, S, K, \mathcal{F}_\phi)$.
    // Compute outcome uncertainty:
    $U_i = \text{Std}(\{R(\tau) \mid \tau \in \mathcal{G}_i\})$.
  end for
  Rank groups by U_i and select top-$p\%$ $\mathcal{X}_{top}$.
  Update π_θ via PPO/GRPO using $\mathcal{X}_{top}$.
end for

4 Experimental Setup

4.1 Environments and Tasks

To ensure reproducibility, we base our experiments on RAGEN (Wang et al., 2025) and choose three diverse multi-turn environments to test agent performance under varying degrees of complexity:

1. **Sokoban**. A logic puzzle where the agent must push boxes into target locations. Feedback is deterministic and instant, allowing to score per-turn rollouts via environment rewards.
2. **FrozenLake**. A stochastic navigation task where the agent must cross a frozen lake while avoiding obstacles (holes). The iced surface is slippery such that the same action can stochastically lead to different states. Rewards are extremely sparse, with a positive signal only upon reaching the goal.
3. **WebShop**. A deterministic e-commerce scenario in which the agent must navigate a website to purchase an item. Rewards are delayed as success is only signaled upon successful purchase, resulting in a long-horizon credit assignment problem.

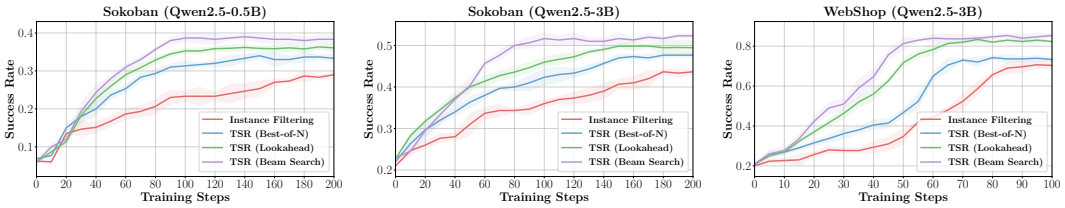
To address sparse or delayed rewards in FrozenLake and WebShop, we guide rollouts with light state-/progress-based scores, while optimizing only the original task reward. We disentangle scoring from search in App. F.6 through reward-only, learned, task-agnostic, and information-matched controls.

4.2 Training and Baselines

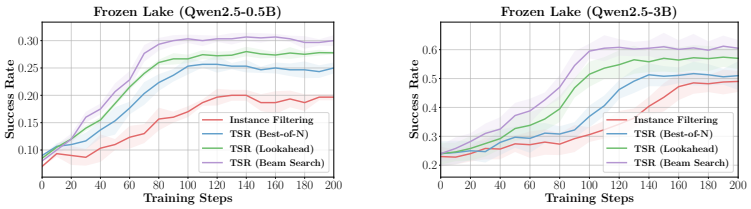
We train Qwen2.5-0.5B and Qwen2.5-3B instruct models on Sokoban and FrozenLake, and only the larger Qwen2.5-3B model on WebShop due to the higher reasoning complexity required for this task. We use PPO for Sokoban and GRPO for FrozenLake and WebShop, as GRPO has been shown to provide more stable learning for these tasks. Following (Yu et al., 2025; Wang et al., 2025), we remove the KL term during optimization and apply asymmetric clipping (via clip-higher). For each batch, we sample $P = 16$ task groups (prompts) with a maximum turn horizon of $K = 5$ for Sokoban/FrozenLake and $K = 10$ for WebShop. Our baseline is instance-level filtering with stochastic rollout sampling using a filtering ratio of $p = 0.25$, similar to default RAGEN. When instantiating TSR with best-of-N, we sample $N = 32$ trajectories and retain the top $L = 16$ trajectories per group. For beam search, we use $M = 4$ and beam width $B = 2$. For lookahead search, we use the same M and B with depth $D = 2$. We selected these parameters after several ablations on N, M, B, and D (see App. F). Additional details on training are provided in App. E.

4.3 Evaluation and Metrics

Across all three tasks, we report success rate as final task accuracy, evaluated on a held-out validation set of 256 fixed prompts per environment. For inference, we use temperature $T = 0.5$ and truncate episodes after 5 turns for Sokoban and FrozenLake and 10 turns for WebShop. Besides success rate (an indicator for *task completion*), we also report rollout entropy (an indicator for diversity, i.e., *exploration*) and average reward distribution for *exploitation*. Training stability is analyzed via gradient norm statistics, where sudden spikes indicate potential mode collapse. We additionally report average response length (*reasoning verbosity*) and the average number of interaction turns (a proxy for *interaction cost and task-completion latency*). Further details on evaluations are given in App. E.



(a) Sokoban (Qwen2.5-0.5B) (b) Sokoban (Qwen2.5-3B) (c) WebShop (Qwen2.5-3B)



(d) FrozenLake (Qwen2.5-0.5B) (e) FrozenLake (Qwen2.5-3B)

Figure 3: **Success Rate Plots.** Comparison of TSR variants (Best-of-N, Lookahead, Beam Search) against the Instance Filtering baseline. Shaded regions show standard deviation across 3 runs.

Table 1: **Results (Sokoban & FrozenLake).** We report Success Rate (↑; mean ± std), Average Response Length in tokens (↓), and Average Interaction Turns (↓) on the held-out validation set.

Task	Method	Qwen2.5-0.5B			Qwen2.5-3B		
		Success Rate (↑)	Resp. Len (↓)	Turns (↓)	Success Rate (↑)	Resp. Len (↓)	Turns (↓)
Sokoban	Base Model	8.9	293	4.7	16.0	272	4.6
	Instance Filtering	29.0 ± 1.7	105	4.4	43.7 ± 1.6	161	4.1
	TSR (Best-of-N)	33.3 ± 1.5	103	4.3	47.7 ± 1.5	165	4.0
	TSR (Lookahead)	36.1 ± 1.4	101	4.0	49.5 ± 1.4	158	3.8
	TSR (Beam Search)	**38.3 ± 1.3**	**98**	**3.8**	**52.3 ± 1.3**	**152**	**3.6**
FrozenLake	Base Model	6.4	276	4.6	12.7	225	3.9
	Instance Filtering	19.7 ± 1.8	157	4.1	48.7 ± 1.9	174	3.6
	TSR (Best-of-N)	25.0 ± 1.6	182	4.2	51.0 ± 1.6	161	3.5
	TSR (Lookahead)	27.8 ± 1.5	168	3.8	57.0 ± 1.4	135	3.3
	TSR (Beam Search)	**30.0 ± 1.4**	**152**	**3.5**	**60.7 ± 1.2**	**98**	**3.1**

5 Results and Discussion

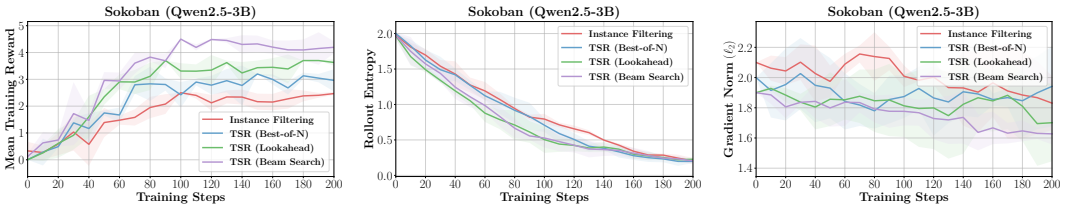
We present key results and statistics assessing final task performance, training stability, and scaling. Further ablations, robustness analyses, and evaluations on other agentic tasks are provided in App. F.

5.1 TSR Performance Comparison and Success Rates

We present success rate across training steps for all agent tasks in Fig. 3 and summarize overall task performance (final success rate) in Table 1 for Sokoban and FrozenLake, and in Table 2 for WebShop.

Across all tasks and models, TSR methods consistently outperform the instance filtering baseline, which only promotes task-level diversity but does not optimize rollouts at the per-turn level. For the 3B model, TSR (Beam Search) achieves a success rate of 52.3% on Sokoban and 60.7% on FrozenLake, improvements of 8.6 and 12 pp over instance filtering. Similarly, TSR (Lookahead) and TSR (Best-of-N) achieve improvements ranging between 2–8 pp. The largest gains are on WebShop, where best-of-N, lookahead, and beam search achieve performance improvements of 3, 12, and 15 pp, respectively, demonstrating that TSR yields strong benefits in this complex long-horizon setting.

Among TSR variants, beam search consistently achieves the strongest performance and converges faster (see Fig. 3), followed by lookahead and best-of-N, which provide smaller gains. Specifically,



(a) Average Reward (Exploitation) (b) Rollout Entropy (Exploration) (c) Gradient Norm (Stability)

Figure 4: **Exploitation, Exploration, and Stability Metrics for Sokoban (Qwen2.5-3B). (a)** TSR achieves higher average rewards, indicating improved exploitation from higher-quality rollouts. **(b)** Rollout entropy decreases smoothly over training, suggesting sustained exploration, followed by policy consolidation. **(c)** Gradient norms remain stable and free of large spikes across TSR variants.

beam search maintains multiple partial trajectories, enabling recovery from early irreversible mistakes and providing robustness to environment stochasticity, particularly in FrozenLake and WebShop.

Lookahead commits to a single action after evaluating short-horizon continuations. While effective in deterministic settings like Sokoban, its gains diminish as search depth increases due to higher cost of evaluating deeper subtrees. Best-of-N improves training by selecting higher-quality *completed* trajectories, i.e., selection occurs only *after* full rollouts are generated. As a result, it acts as an oversampled variant of instance filtering, increasing the chance of "luckier" rollouts without actively steering intermediate decisions.

Table 2: **Results (WebShop, Qwen2.5-3B).** Success Rate (↑; mean ± std), Average Response Length (↓), and Average Interaction Turns (↓) on validation set.

Method	Qwen2.5-3B		
	Success Rate (↑)	Resp. Len (↓)	Turns (↓)
Base Model	3.0	747	7.7
Instance Filtering	70.3 ± 1.8	519	6.8
TSR (Best-of-N)	73.3 ± 1.5	504	6.5
TSR (Lookahead)	82.3 ± 1.3	475	6.1
TSR (Beam Search)	$\mathbf{85.3 \pm 1.1}$	**453**	**5.8**

5.2 Exploitation, Exploration, and Training Stability

We analyze exploitation, exploration, and training stability using diagnostic statistics, shown exemplarily in Fig. 4 for the Sokoban task with the Qwen2.5-3B model.

For exploitation, we analyze average training rewards in Fig. 4a. Across all TSR variants, average rewards increase steadily over training, indicating effective learning and policy improvement. The trends are further consistent with the overall task performance: beam search has on average higher rewards, followed by lookahead and best-of-N. This indicates that TSR shifts the exploitation bias toward higher rewards, resulting in stronger performance due to finding better trajectories.

For exploration, we analyze rollout entropy in Fig. 4b, which measures the uncertainty of the model's token predictions. Across all methods, entropy decays smoothly over training, indicating that the agent is sufficiently exploring different strategies with uncertain outcomes in the beginning before converging to a more concentrated, exploitative behavior with a specific reasoning path. Importantly, TSR variants exhibit a similar entropy decay profile to the instance filtering baseline, suggesting that search-guided rollout does not alter or destabilize the exploration–exploitation dynamics.

Finally, we analyze gradient norms in Fig. 4c as a measure of stability. Sharp spikes in gradient norms are indicative of the "Echo Trap" and can lead to irreversible instability and mode collapse (Wang et al., 2025; Xi et al., 2025a). Across all methods, gradient norms remain without spikes.

5.3 Inference Efficiency

In addition to final success rates, Tables 1 and 2 report average response length and interaction turns to assess whether train-time trajectory search can improve solution quality, token efficiency, and interaction cost at inference time. Across experiments, TSR consistently reduces both metrics, indicating improved decision-making at test time. In particular, beam search produces the shortest responses and fewest turns on average, with lookahead achieving comparable reductions. Overall, this suggests that TSR distills more concise interaction strategies into the policy without requiring search at deployment. We substantiate these results with qualitative examples for WebShop in App. F.1.

Table 3: **Rollout-Budget-Matched Baselines (FrozenLake, Qwen2.5-3B).** Instance filtering plateaus, TSR uses compute more effectively.

Method	Rel. Rollout Budget	Success Rate
Instance Filtering	1.0×	48.7 ± 1.9
Instance Filtering	2.0×	49.5 ± 1.7
Instance Filtering	4.0×	49.8 ± 1.6
TSR (Beam Search)	2.0×	$\mathbf{60.7 \pm 1.2}$

Table 4: **Scaling Effects (Sokoban, Qwen2.5-3B).** Performance gains saturate as search budgets increase. $B = 2$ offers the best trade-off.

Method	Search Budget	Success Rate	Δ
TSR (Beam Search)	$M = 2, B = 1$	50.6 ± 1.4	–
	$M = 4, B = 2$	$\mathbf{52.3 \pm 1.3}$	+1.7
	$M = 6, B = 2$	53.4 ± 1.2	+1.1

5.4 Rollout-Budget-Matched Baselines

Since TSR spends additional compute during rollout generation, we evaluate whether its gains can be matched by simply scaling naive rollout sampling. Specifically, we keep the number of task groups $P = 16$, the number of retained trajectories $L_{\text{train}} = 16$, the downstream optimizer, and the evaluation protocol fixed, while increasing the number of independently generated trajectories L_{gen}. For Qwen2.5-3B on FrozenLake, Table 3 shows that increasing L_{gen} from 16 to 32 and 64, corresponding to $1\times$, $2\times$, and $4\times$ rollout-generation budgets, improves success only from 48.7% to 49.5% and 49.8%, respectively. In contrast, TSR (Beam Search) reaches 60.7% under the matched $2\times$ rollout-generation budget. This substantiates that TSR's gains are not explained by additional sampling compute alone, but by allocating that compute structurally to search over intermediate decisions. A more detailed rollout budget accounting is provided in App. F.2.

5.5 Search Budget, Performance, and Distribution Shift

We further study how increasing train-time search budgets affects final performance. As shown in Table 4, we observe diminishing marginal returns. Performance largely saturates as M increases, with differences between larger sampling budgets often within the observed run-to-run variation. This justifies our choice of moderate budgets (e.g., $B = 2$, $M = 4$) for our experiments (see App. F.3).

Table 5: **Distribution Shift Analysis (FrozenLake, Qwen2.5-3B).** Moderate beam widths are policy-proximal, while excessive search increases KL, degrading performance.

B	Avg. KL / Step	Success Rate
2	~ 0.02	$\mathbf{60.7 \pm 1.2}$
5	~ 0.05	61.3 ± 1.1
10	~ 0.14	60.9 ± 1.4
20	~ 0.31	53.8 ± 1.8
30	~ 0.48	45.1 ± 2.1

Increasing search budgets also affects the distribution shift between TSR's search-guided rollout distribution $\mu_{\theta,\phi}$ and the current policy π_θ. In general, PPO and GRPO accommodate this shift through clipping mechanisms, which enforce a trust region on each update. To characterize when this shift becomes harmful, we perform an extreme beam width ablation and report the average KL divergence in Table 5. At moderate beam widths ($B \leq 5$), KL divergence remains low ($\lesssim 0.05$) and performance remains high. At $B = 10$, the distribution shift becomes more pronounced with only marginal gains, while wider beams degrade learning. This shows that harmful distribution shift arises only far outside moderate beam widths. Thus, in practical operating regimes, TSR behaves as a policy-proximal search procedure (see App. F.4).

6 Conclusion

In this work, we introduced Trajectory-Search Rollouts (TSR), a train-time rollout generation framework that adapts search methods, such as beam search, lookahead, and best-of-N, to enhance the training data quality in multi-turn agentic RL. By exploring and pruning trajectories at the per-turn level, TSR changes only the rollout distribution and thus remains compatible with standard policy gradient optimizers. Across Sokoban, FrozenLake, and WebShop tasks, TSR improves final task accuracy by up to 15 percentage points, while producing shorter responses and fewer interaction turns at inference time. Our analysis further shows that (1) TSR improves the exploitation–exploration trade-off by steering rollouts toward higher-signal trajectories without destabilizing policy learning, (2) TSR's gains stem from structured search rather than additional rollout budget alone, and (3) TSR remains policy-proximal in practical operating regimes, while harmful distribution shift appears only under excessive search. Thus, TSR demonstrates that moving search to train-time rollout construction can substantially improve performance, providing a modular way to trade additional train-time rollout compute for stronger deployment-time policies. We discuss limitations and broader impact in App. G.

Reproducibility Statement

All environments and rollout scoring signals, including their hyperparameters, are described in App. D. App. E specifies the models, optimization and search hyperparameters, the rollout-budget accounting for each search strategy, the evaluation protocol, and the hardware used. Implementation details for all baselines, including our adaptations of ARPO and GiGPO, are given in App. F.8. All reported results are averaged over three independent runs. Our implementation builds on the publicly available RAGEN framework. The complete code and configuration files are included as anonymous supplementary material and will be released publicly upon acceptance.

References

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. WebGPT: Browser-Assisted Question-Answering with Human Feedback, 2022. URL https://arxiv.org/abs/2112.09332.

Yifei Zhou, Andrea Zanette, Jiayi Pan, Sergey Levine, and Aviral Kumar. ArCHer: Training Language Model Agents via Hierarchical Multi-Turn RL, 2024. URL https://arxiv.org/abs/2402.19446.

Yifei Zhou, Song Jiang, Yuandong Tian, Jason Weston, Sergey Levine, Sainbayar Sukhbaatar, and Xian Li. SWEET-RL: Training Multi-Turn LLM Agents on Collaborative Reasoning Tasks, 2025. URL https://arxiv.org/abs/2503.15478.

Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Xing Jin, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, Eli Gottlieb, Yiping Lu, Kyunghyun Cho, Jiajun Wu, Li Fei-Fei, Lijuan Wang, Yejin Choi, and Manling Li. RAGEN: Understanding Self-Evolution in LLM Agents via Multi-Turn Reinforcement Learning, 2025. URL https://arxiv.org/abs/2504.20073.

Xufang Luo, Yuge Zhang, Zhiyuan He, Zilong Wang, Siyun Zhao, Dongsheng Li, Luna K. Qiu, and Yuqing Yang. Agent Lightning: Train ANY AI Agents with Reinforcement Learning, 2025. URL https://arxiv.org/abs/2508.03680.

Zhiheng Xi, Jixuan Huang, Chenyang Liao, Baodai Huang, Honglin Guo, Jiaqi Liu, Rui Zheng, Junjie Ye, Jiazheng Zhang, Wenxiang Chen, Wei He, Yiwen Ding, Guanyu Li, Zehui Chen, Zhengyin Du, Xuesong Yao, Yufei Xu, Jiecao Chen, Tao Gui, Zuxuan Wu, Qi Zhang, Xuanjing Huang, and Yu-Gang Jiang. AgentGym-RL: Training LLM Agents for Long-Horizon Decision Making through Multi-Turn Reinforcement Learning, 2025a. URL https://arxiv.org/abs/2509.08755.

Shiyi Cao, Dacheng Li, Fangzhou Zhao, Shuo Yuan, Sumanth R. Hegde, Connor Chen, Charlie Ruan, Tyler Griggs, Shu Liu, Eric Tang, Richard Liaw, Philipp Moritz, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. SkyRL-Agent: Efficient RL Training for Multi-turn LLM Agent, 2025. URL https://arxiv.org/abs/2511.16108.

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, 2025. URL https://arxiv.org/abs/2502.01456.

Zilin Zhu, Chengxing Xie, Xin Lv, and slime Contributors. slime: An LLM post-training framework for RL Scaling. https://github.com/THUDM/slime, 2025a. GitHub repository. Corresponding author: Xin Lv.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models, 2024. URL https://arxiv.org/abs/2402.03300.

Quan Wei, Siliang Zeng, Chenliang Li, William Brown, Oana Frunza, Wei Deng, Anderson Schneider, Yuriy Nevmyvaka, Yang Katie Zhao, Alfredo Garcia, and Mingyi Hong. Reinforcing Multi-Turn Reasoning in LLM Agents via Turn-Level Reward Design, 2025. URL https://arxiv.org/abs/2505.11821.

Vaishnavi Shrivastava, Ahmed Awadallah, Vidhisha Balachandran, Shivam Garg, Harkirat Behl, and Dimitris Papailiopoulos. Sample More to Think Less: Group Filtered Policy Optimization for Concise Reasoning, 2025. URL https://arxiv.org/abs/2508.09726.

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, 2023. URL https://arxiv.org/abs/2304.06767.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters, 2024. URL https://arxiv.org/abs/2408.03314.

Thomas Anthony, Zheng Tian, and David Barber. Thinking Fast and Slow with Deep Learning and Tree Search, 2017. URL https://arxiv.org/abs/1705.08439.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 Technical Report, 2025. URL https://arxiv.org/abs/2412.15115.

Théophane Weber, Sébastien Racanière, David P. Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adria Puigdomènech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, Razvan Pascanu, Peter Battaglia, Demis Hassabis, David Silver, and Daan Wierstra. Imagination-Augmented Agents for Deep Reinforcement Learning, 2018. URL https://arxiv.org/abs/1707.06203.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym, 2016. URL https://arxiv.org/abs/1606.01540.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. WebShop: Towards Scalable Real-World Web Interaction with Grounded Language Agents, 2023a. URL https://arxiv.org/abs/2207.01206.

Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy Gradient Methods for Reinforcement Learning with Function Approximation. *Advances in neural information processing systems*, 12, 1999.

John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-Dimensional Continuous Control Using Generalized Advantage Estimation. *arXiv preprint arXiv:1506.02438*, 2015.

Youssef Mroueh, Nicolas Dupuis, Brian Belgodere, Apoorva Nitsure, Mattia Rigotti, Kristjan Greenewald, Jiri Navratil, Jerret Ross, and Jesus Rios. Revisiting Group Relative Policy Optimization: Insights into On-Policy and Off-Policy Training. *arXiv preprint arXiv:2505.22257*, 2025.

Oleksii Ignatenko and Ruslan Pravosud. Solving Sokoban Game with a Heuristic for Avoiding Dead-End States. In *ICTERI*, 2023. https://doi.org/10.1007/978-3-031-48325-7_4.

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. *arXiv preprint arXiv:2305.10601*, 2023b.

Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. ReST-MCTS*: LLM Self-Training via Process Reward Guided Tree Search. *arXiv preprint arXiv:2406.03816*, 2024.

Burr Settles. From Theories to Queries: Active Learning in Practice. In *Active Learning and Experimental Design Workshop in Conjunction with AISTATS 2010*, pages 1–18. JMLR Workshop and Conference Proceedings, 2011.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, Yonghui Wu, and Mingxuan Wang. DAPO: An Open-Source LLM Reinforcement Learning System at Scale, 2025. URL https://arxiv.org/abs/2503.14476.

Zhiheng Xi, Yiwen Ding, Wenxiang Chen, Boyang Hong, Honglin Guo, Junzhe Wang, Dingwen Yang, Chenyang Liao, Xin Guo, Wei He, Songyang Gao, Lu Chen, Rui Zheng, Yicheng Zou, Tao Gui, Qi Zhang, Xipeng Qiu, Xuanjing Huang, Zuxuan Wu, and Yu-Gang Jiang. AgentGym: Evolving Large Language Model-based Agents across Diverse Environments, 2024. URL https://arxiv.org/abs/2406.04151.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training Language Models to Follow Instructions with Human Feedback, 2022. URL https://arxiv.org/abs/2203.02155.

Lin Gui, Cristina Gârbacea, and Victor Veitch. BoNBoN Alignment for Large Language Models and the Sweetness of Best-of-n Sampling, 2024. URL https://arxiv.org/abs/2406.00832.

Huaijie Wang, Shibo Hao, Hanze Dong, Shenao Zhang, Yilin Bao, Ziran Yang, and Yi Wu. Offline Reinforcement Learning for LLM Multi-Step Reasoning, 2024. URL https://arxiv.org/abs/2412.16145.

Hazim Alzorgan and Abolfazl Razi. Monte Carlo Beam Search for Actor-Critic Reinforcement Learning in Continuous Control, 2025. URL https://arxiv.org/abs/2505.09029.

Randy Lefebvre and Audrey Durand. On Shallow Planning under Partial Observability, 2025. URL https://arxiv.org/abs/2407.15820.

King Zhu, Hanhao Li, Siwei Wu, Tianshun Xing, Dehua Ma, Xiangru Tang, Minghao Liu, Jian Yang, Jiaheng Liu, Yuchen Eleanor Jiang, Changwang Zhang, Chenghua Lin, Jun Wang, Ge Zhang, and Wangchunshu Zhou. Scaling Test-time Compute for LLM Agents, 2025b. URL https://arxiv.org/abs/2506.12928.

Junhong Shen, Hao Bai, Lunjun Zhang, Yifei Zhou, Amrith Setlur, Shengbang Tong, Diego Caples, Nan Jiang, Tong Zhang, Ameet Talwalkar, and Aviral Kumar. Thinking vs. Doing: Agents that Reason by Scaling Test-Time Interaction, 2025. URL https://arxiv.org/abs/2506.07976.

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. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm, 2017. URL https://arxiv.org/abs/1712.01815.

Yuxiang Ji, Ziyu Ma, Yong Wang, Guanhua Chen, Xiangxiang Chu, and Liaoni Wu. Tree Search for LLM Agent Reinforcement Learning, 2026. URL https://arxiv.org/abs/2509.21240.

Guanting Dong, Hangyu Mao, Kai Ma, Licheng Bao, Yifei Chen, Zhongyuan Wang, Zhongxia Chen, Jiazhen Du, Huiyang Wang, Fuzheng Zhang, Guorui Zhou, Yutao Zhu, Ji-Rong Wen, and Zhicheng Dou. Agentic Reinforced Policy Optimization, 2025a. URL https://arxiv.org/abs/2507.19849.

Guanting Dong, Licheng Bao, Zhongyuan Wang, Kangzhi Zhao, Xiaoxi Li, Jiajie Jin, Jinghan Yang, Hangyu Mao, Fuzheng Zhang, Kun Gai, Guorui Zhou, Yutao Zhu, Ji-Rong Wen, and Zhicheng Dou. Agentic Entropy-Balanced Policy Optimization, 2025b. URL https://arxiv.org/abs/2510.14545.

Shangyu Xing, Siyuan Wang, Chenyuan Yang, Xinyu Dai, and Xiang Ren. Lookahead Tree-Based Rollouts for Enhanced Trajectory-Level Exploration in Reinforcement Learning with Verifiable Rewards, 2026. URL https://arxiv.org/abs/2510.24302.

Xintong Li, Sha Li, Yuwei Zhang, Changlong Yu, Rongmei Lin, Hongye Jin, Shuyi Guan, Xin Liu, Linwei Li, Qingyu Yin, and Jingbo Shang. Process Reward Informed Tree Rollout for Effective Multi-Turn RL, 2026. URL https://arxiv.org/abs/2607.15610.

Zhiheng Xi, Chenyang Liao, Guanyu Li, Yajie Yang, Wenxiang Chen, Zhihao Zhang, Binghai Wang, Senjie Jin, Yuhao Zhou, Jian Guan, Wei Wu, Tao Ji, Tao Gui, Qi Zhang, and Xuanjing Huang. AgentPRM: Process Reward Models for LLM Agents via Step-Wise Promise and Progress, 2025b. URL https://arxiv.org/abs/2511.08325.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-Group Policy Optimization for LLM Agent Training, 2025. URL https://arxiv.org/abs/2505.10978.

Leitian Tao, Baolin Peng, Wenlin Yao, Tao Ge, Hao Cheng, Mike Hang Wang, Jianfeng Gao, and Sharon Li. TRACE: Turn-level Reward Assignment via Credit Estimation for Long-Horizon Agents, 2026. URL https://arxiv.org/abs/2607.13988.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-Dimensional Continuous Control Using Generalized Advantage Estimation, 2018. URL https://arxiv.org/abs/1506.02438.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning Text and Embodied Environments for Interactive Learning, 2021. URL https://arxiv.org/abs/2010.03768.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A Controllable World of Apps and People for Benchmarking Interactive Coding Agents, 2024. URL https://arxiv.org/abs/2407.18901.

A Related Works

We present a short overview of relevant related works.

Multi-Turn Agentic RL Frameworks. Recent works have developed practical frameworks for training multi-turn LLM agents across diverse environments. Agent Lightning (Luo et al., 2025) and AgentGym-RL (Xi et al., 2025a) provide decoupled training systems and modular pipelines, building on AgentGym (Xi et al., 2024), which focuses primarily on environment diversity and data collection. SkyRL (Cao et al., 2025) offers a full-stack library for tool-use agents with asynchronous weight updates to maximize throughput. Similarly, slime (Zhu et al., 2025a) decouples training from rollouts by connecting Megatron with SGLang, specifically targeting scaling of multi-turn RL. RAGEN (Wang et al., 2025) provides a trajectory-level RL formulation and documents instability patterns such as the Echo Trap, while their StarPO-S method mitigates collapse via instance filtering and gradient shaping. TSR is complementary to these systems. It modifies the rollout generator through structured trajectory search while leaving the downstream optimizer, loss, and training framework unchanged. We adopt instance filtering from RAGEN as a separate stabilizer in our experiments, while our main contribution is search-guided rollout generation within the training loop.

Test-Time Scaling and Rollout Search. Prior work has investigated test-time scaling strategies such as Best-of-N sampling (Ouyang et al., 2022; Gui et al., 2024), beam search (Wang et al., 2024; Alzorgan and Razi, 2025), and shallow lookahead (Lefebvre and Durand, 2025). These approaches allocate additional compute at inference time to improve solution quality. TSR transfers this principle to training-time rollouts, where additional search is performed during training and distilled into a policy that requires no search at deployment. Recent work has also extended test-time scaling to multi-turn agents by balancing search breadth against environmental-feedback depth (Zhu et al., 2025b) or scaling inference-time interaction and backtracking (Shen et al., 2025). Unlike these approaches, TSR places the additional search within rollout construction during RL training, allowing the resulting behavior to be internalized by the policy.

Search-Guided Policy Improvement and Expert Iteration. TSR is conceptually related to Expert Iteration (Anthony et al., 2017), where tree search acts as a slow expert that improves upon the current apprentice policy and the neural network subsequently generalizes the improved plans back into a fast policy. This search-and-distillation view also underlies AlphaZero-style (Silver et al., 2017) policy improvement approaches. TSR follows the same high-level principle for multi-turn LLM agentic RL, where search improves the rollout distribution during training and PPO/GRPO distill the resulting higher-quality behavior into the policy. Unlike classical Expert Iteration, TSR does not rely on supervised imitation targets from a separate expert but instead uses search-guided trajectories directly within standard policy-gradient training.

Rejection-Sampling-Style Selection. Several training methods perform response- or trajectory-level filtering after samples have been generated. For example, GFPO (Shrivastava et al., 2025) filters responses by properties such as length or reward-per-token within GRPO-style updates, while RAFT-style methods (Dong et al., 2023) train on positively rewarded samples. TSR differs in that it targets interactive multi-turn agent RL and modifies trajectory construction before completion through per-turn branching, scoring, and pruning, rather than only filtering completed responses. We view these approaches as complementary and compare TSR against search-free selection baselines under matched rollout-generation budgets.

Tree-Structured and Adaptive Rollouts. Several recent methods replace independent chain sampling with tree-structured rollouts, but they use the tree for different purposes. Tree-GRPO (Ji et al., 2026) expands randomly selected agent-step nodes to obtain more trajectories per token and tool-call budget, and derives intra- and inter-tree advantages from outcome rewards, so the tree serves sample efficiency and credit assignment. ARPO (Dong et al., 2025a) and AEPO (Dong et al., 2025b) branch at high-entropy tool-call steps to broaden exploration, coupled with modified advantage estimation and, for AEPO, a modified clipping rule. LATR (Xing et al., 2026) branches at uncertain tokens and prunes near-duplicate continuations to increase diversity in single-turn RLVR. In these methods, branching is driven by randomness or policy-intrinsic uncertainty, and pruning, where present, targets redundancy rather than return. Concurrent work, PATR (Li et al., 2026), scores partial trajectories every K turns and gives promising branches more continuations, while all completed and pruned branches still enter the rollout group. In all of these approaches, the tree shapes where rollout budget is spent but not which trajectories the policy learns from. TSR instead treats rollout generation as a policy-improvement step. Candidate actions proposed by π_θ are executed and scored at every turn, and search selects the trajectories that enter the update, so the policy learns from higher-return behavior than it would produce unaided. Compared with PATR, TSR evaluates every decision point rather than periodic checkpoints, can score candidates by their downstream consequences through lookahead or Monte Carlo continuations, and instantiates a family of search strategies under a single rollout interface. This applies the Expert Iteration principle inside on-policy agentic RL and raises a question that allocation- and exploration-oriented methods do not face, namely how far selection can shift the data away from π_θ before clipped updates stop benefiting. While PATR notes that adaptive allocation biases the rollout group without quantifying this effect, we characterize the boundary explicitly in Sec. 5.5. Because TSR changes only which trajectories are trained on, it is complementary to these rollout schemes.

Turn-Level Credit Assignment. Recent work addresses sparse trajectory rewards by improving credit assignment within multi-turn trajectories. MT-PPO/MT-GRPO (Wei et al., 2025) introduce turn-level reward designs to provide denser supervision for multi-turn reasoning, while AgentPRM (Xi

et al., 2025b) uses process reward models to provide step-wise evaluations for long-horizon tasks. GiGPO (Feng et al., 2025) introduces a group-in-group advantage structure that combines episode-level relative advantages with step-level groups formed from repeated anchor states, enabling fine-grained credit assignment without additional rollouts. TRACE (Tao et al., 2026) similarly targets long-horizon credit assignment by deriving dense turn-level rewards from changes in reference-model state values. These methods primarily refine the learning signal or advantage estimator, whereas TSR focuses on improving the rollout distribution before the policy update. We evaluate this distinction directly in App. F.8, where we compare TSR against GiGPO and additionally study their composition.

B Policy Gradient Methods in Multi-Turn RL

The core idea of policy gradient methods (Sutton et al., 1999) is to perform gradient ascent according to an objective $J(\theta)$, which typically represents the expected cumulative reward that the agent receives when following the policy π_θ and interacting with the environment.

B.1 PPO and GRPO for Multi-Turn RL

We focus on two widely adopted approaches and adapt them to the multi-turn setting.

Proximal Policy Optimization (PPO). PPO (Schulman et al., 2017) stabilizes training by limiting the magnitude of policy updates via a clipped surrogate objective:

$$J_{\text{PPO}}(\theta) = \mathbb{E}_{\tau\sim\pi_{\theta_{\text{old}}}}\left[\frac{1}{K}\sum_{t=0}^{K-1}\min\left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right], \tag{5}$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|\tau_{<t})}{\pi_{\theta_{\text{old}}}(a_t|\tau_{<t})}$ is the turn-level importance-sampling ratio between the updated and behavior policy. In standard PPO, $\hat{A}_t$ is typically computed using Generalized Advantage Estimation (GAE) (Schulman et al., 2018), which relies on a learned value function (critic).

Group Relative Policy Optimization (GRPO). Learning a separate value function can still be computationally expensive and unstable. GRPO (Shao et al., 2024) avoids an explicit critic by estimating advantages using relative comparisons. Given a group of G trajectories with returns $\{R_1, \ldots, R_G\}$ sampled for the same task, the advantage for the i-th trajectory is computed as:

$$\hat{A}_i = \frac{R_i - \text{mean}(R_1, \ldots, R_G)}{\text{std}(R_1, \ldots, R_G) + \epsilon}. \tag{6}$$

GRPO then optimizes a PPO-style clipped objective averaged over the group:

$$J_{\text{GRPO}}(\theta) = \mathbb{E}_{\{\tau_i\}_{i=1}^{G}\sim\pi_{\theta_{\text{old}}}}\left[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{K_i}\sum_{t=0}^{K_i-1}\min\left(r_{i,t}(\theta)\,\hat{A}_i,\ \text{clip}(r_{i,t}(\theta), 1-\epsilon, 1+\epsilon)\,\hat{A}_i\right)\right], \tag{7}$$

where K_i denotes the number of turns in τ_i, and $r_{i,t}(\theta) = \frac{\pi_\theta(a_{i,t}|\tau_{i,<t})}{\pi_{\theta_{\text{old}}}(a_{i,t}|\tau_{i,<t})}$ is the per-turn importance-sampling ratio, defined analogously to PPO.

B.2 Search-Guided Rollouts and Policy Gradient Compatibility

TSR modifies the data generation process used during rollout collection. Instead of drawing complete trajectories directly from the current policy π_θ, TSR samples candidate actions from π_θ and applies a search strategy $\mathcal{F}_\phi$ guided by a scoring function S. This induces a search-guided rollout distribution

$$\mu_{\theta,\phi}(\tau \mid u; S), \tag{8}$$

which generally differs from the naive on-policy rollout distribution $\pi_\theta(\tau \mid u)$. Thus, TSR should not be interpreted as an exact unbiased Monte Carlo estimator of the vanilla on-policy policy gradient objective. Instead, TSR is best viewed as a policy-proximal, search-guided policy-improvement procedure. This perspective is closely related to Expert Iteration and rejection-sampling-style fine-tuning, where a search or filtering procedure acts as an improvement operator over the current

policy, producing higher-quality behavior than naive sampling. The policy is then updated to distill this improved behavior. Similarly, TSR uses π_θ to propose candidate actions, search to reweight trajectories toward higher-scoring prefixes, and PPO/GRPO to update the policy using the original task reward. Importantly, TSR does not train directly on arbitrary off-policy data. Instead, all candidate actions are sampled from π_θ and search only selects among these policy-generated candidates.

For GRPO-based experiments, this selection also changes the within-group return distribution used for group-relative advantage estimation. This is most explicit for Best-of-N, where advantages are computed over the L_{train} retained trajectories rather than all N generated candidates. The selection therefore intentionally removes part of the low-return tail as part of the rollout-improvement operator. Relative positive and negative advantages remain whenever the retained trajectories exhibit non-zero return variance, while groups with identical retained returns provide no group-relative learning signal and receive low uncertainty under the subsequent instance-filtering step.

TSR and instance filtering are therefore applied sequentially rather than independently. For each task group, instance-filtering uncertainty is computed over the L_{train} trajectories returned by the rollout generator, i.e., over the post-search return distribution. Search can consequently change within-group return variance and thereby affect which task groups are prioritized for policy updates. We interpret instance filtering in this setting as measuring the residual outcome uncertainty after rollout improvement. Thus, groups whose post-search outcomes remain heterogeneous are prioritized, whereas groups that become uniformly successful or unsuccessful receive lower priority. The same post-generation filtering rule is applied consistently across methods, although we do not separately disentangle this interaction in the present study.

Therefore, even though $\mu_{\theta,\phi}$ is not identical to π_θ, the practical question is whether the induced shift remains small enough for clipped policy gradient updates to learn effectively from the resulting trajectories. To this end, PPO and GRPO optimize clipped surrogate objectives whose probability ratios limit the effect of actions that become too unlikely under the updated policy. As shown in Sec. 5.5 of the main body (see also App. F.4), in moderate search regimes, the selected trajectories remain policy-proximal while improving the quality of the rollout data. However, if the search budget is made too large, $\mu_{\theta,\phi}$ can move too far from π_θ, limiting the effective learning signal under clipping and reducing the usefulness of the rollout buffer. This is precisely what we observe empirically: increasing the beam width initially improves performance while the average KL divergence remains small, whereas excessive beam widths substantially increase KL divergence and degrade performance. These results support the view that TSR is effective when used as a moderate, policy-proximal search procedure rather than as an unconstrained search oracle.

C Trajectory Search Rollout Search Strategies

We instantiate TSR with three concrete search strategies inspired by test-time scaling (Snell et al., 2024): best-of-N, beam search, and shallow lookahead. All three methods sample candidate actions from the current policy π_θ and use a scoring function S to retain higher-scoring trajectories for policy optimization. The examples below use Sokoban for explanation, where early box pushes can create irreversible deadlocks. However, the same strategies apply generally to any multi-turn environment in which candidate actions can be evaluated using available rewards or state-based feedback. Fig. 5 provides a conceptual overview of how best-of-N, beam search, and shallow lookahead modify rollout generation in multi-turn RL.

Best-of-N. From a given initial state, the policy samples N complete rollouts independently. In Sokoban, some sampled trajectories may commit early to irreversible moves, such as pushing a box into a wall or corner, and therefore terminate with low return (see Fig. 1). Other trajectories may preserve maneuverability and eventually solve the puzzle. Best-of-N scores the completed rollouts and retains the top-L highest-scoring trajectories for policy optimization. Because selection occurs only after full trajectories are generated, best-of-N improves rollout quality but does not actively steer intermediate decisions.

Beam Search. Beam search performs selection at the level of partial trajectories. At each turn, the policy samples candidate actions for each active prefix and TSR scores the resulting environment outcomes using S. The search then retains the top-B prefixes and prunes the rest. In Sokoban, this

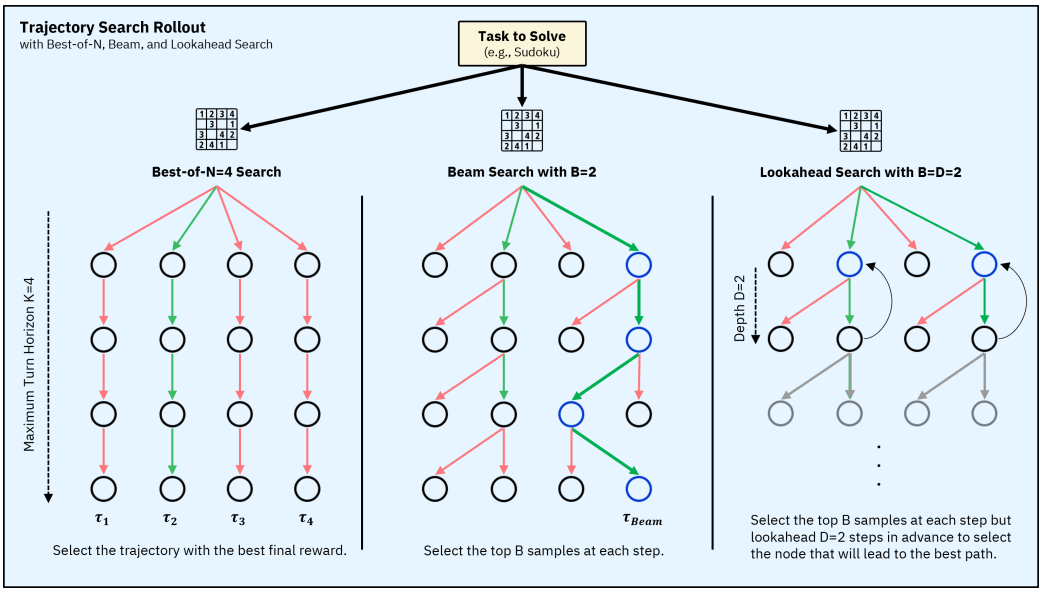


Figure 5: Illustration of TSR-adapted tree-search rollout generation for multi-turn RL training with best-of-N, beam search, and shallow lookahead search strategies.

allows beam search to discard prefixes that move toward dead-end configurations while preserving alternative branches that keep future pushes feasible. Thus, beam search can recover from locally attractive but irreversible decisions by maintaining multiple candidate futures in parallel.

Shallow Lookahead. Shallow lookahead evaluates each candidate action by simulating a short continuation of depth $D \ll K$ before committing. In Sokoban, an action whose immediate effect appears useful may quickly lead to blocked or low-mobility states after a few future pushes. Lookahead avoids such actions when their short-horizon continuations score poorly, favoring actions that preserve access and future maneuverability. Compared with beam search, shallow lookahead uses a shorter planning horizon and commits to a single action after evaluating the truncated subtree, reducing overhead while still improving over purely greedy-based rollout sampling.

D RL Environments

This section describes the Sokoban, FrozenLake, and WebShop environments used in our experiments. We additionally describe the state-based scoring signals used by TSR for rollout construction in sparse- or delayed-reward settings. Importantly, these signals are used only to guide search during rollout generation, i.e., policy optimization remains based on the original task reward.

D.1 Environment Descriptions

Sokoban. Sokoban (Weber et al., 2018) is a deterministic logic puzzle in which the agent must push boxes onto designated target locations within a grid-based warehouse. The agent navigates the grid and interacts with boxes via orthogonal pushes, but cannot pull boxes or push multiple boxes simultaneously. The objective is to place all boxes on target cells while avoiding irreversible deadlocks, such as pushing boxes into corners or against walls where they can no longer be moved to a target. In our setup (chosen from the RAGEN framework), rewards are provided at each step to capture both accuracy and efficiency: $+1$ per box on target, -1 per box off target, $+10$ on task completion, and -0.1 per action. A sample environment is shown in Fig. 6.

FrozenLake. FrozenLake (Brockman et al., 2016) is a stochastic navigation task in which the agent must cross a grid-based frozen lake from a start state to a goal while avoiding holes. The environment consists of safe frozen tiles, terminal hole states, and a terminal goal state. The task is challenging

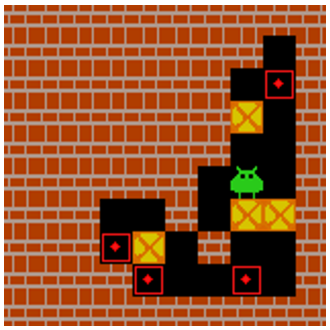


Figure 6: Sokoban.



Figure 7: FrozenLake.

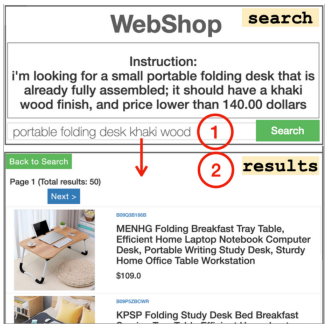


Figure 8: WebShop.

Table 6: **Scoring Signals Used by TSR.** State-based scores guide rollout construction only, whereas PPO/GRPO updates remain based on the original task reward.

Environment	TSR Rollout Scoring Signal	Policy Optimization Reward
Sokoban	Step-level environment reward / box-target progress	Original task reward
FrozenLake	Goal-distance reduction and hole avoidance	Sparse goal-reaching reward
WebShop	Milestone progress and keyword/attribute overlap	Final purchase success reward

because transitions are stochastic such that each action succeeds with probability $1/3$ and deviates perpendicularly with probability $2/3$ due to the slippery surface. Rewards are sparse, with a positive reward of $+1$ only upon reaching the goal and 0 otherwise. A sample environment is shown in Fig. 7.

WebShop. WebShop (Yao et al., 2023a) is a deterministic e-commerce environment in which the agent must find and purchase a product satisfying a natural-language instruction. The environment contains over one million real-world products with structured attributes, descriptions, and prices. The agent interacts with the website through high-level actions such as searching, clicking product links, selecting options, and purchasing. The task requires mapping the instruction's product constraints to available items while filtering irrelevant search results. Rewards are delayed because success is determined only after the agent reaches the final purchase state with the correct item. In our experiments, WebShop uses RAGEN's `webshop-minimal` adaptation. A sample environment is shown in Fig. 8.

D.2 State-Based Scoring Signals for Rollout Construction

In sparse- or delayed-reward environments, the environment reward may be uninformative for ranking candidate actions during intermediate rollout steps. For example, FrozenLake provides a non-zero reward only upon reaching the goal and WebShop provides success feedback only after the final purchase. To guide TSR in such settings, we use lightweight state-based scoring signals that evaluate the outcome of a candidate action after it is executed in the environment (see Table 6). These scores are not optimized as dense rewards by PPO/GRPO. They are only used to rank candidate actions or trajectory prefixes during rollout construction, while policy optimization remains based on the original task reward. Thus, TSR decouples *search guidance* from *policy optimization*: state-based scores help the rollout generator explore promising regions of the trajectory space, whereas the RL update continues to optimize the true task objective. The resulting signals are deliberately simple and are derived from environment state variables already exposed by the simulator. More generally, scores may also be instantiated using other environment metadata, rule-based progress checks, learned value models, or process reward models. Because these scoring signals provide additional training-time information, we explicitly isolate their contribution in App. F.6. Across reward-only, task-agnostic, learned, score-matched, and heuristic-free controls, TSR's structured search provides gains beyond access to the scoring signal itself. In particular, Monte Carlo-backed beam search outperforms Best-of-N using only the original terminal task reward, at additional rollout-generation cost, while an observation-only value model on WebShop shows that hidden simulator state is not required. Thus, auxiliary scoring can strengthen TSR, but is not a prerequisite for effective trajectory search.

FrozenLake. FrozenLake has stochastic transition dynamics and provides a non-zero reward only upon reaching the goal. Directly using the environment reward to rank intermediate candidate actions is therefore ineffective, since most non-terminal actions receive the same reward. We instead use a simple state-based score that balances progress toward the goal with risk avoidance. Let s_t denote the agent's current grid position, g the goal location, and $\mathcal{H}$ the set of hole states. After executing action a_t and observing the resulting state s_{t+1}, we define

$$S_{\text{FL}}(s_t, a_t) = -d(s_{t+1}, g) - \lambda \cdot \mathbb{I}[s_{t+1} \in \mathcal{H}] \,, \tag{9}$$

where $d(\cdot, \cdot)$ denotes the Manhattan distance, $\mathbb{I}[\cdot]$ is the indicator function, and $\lambda > 0$ controls the penalty for entering a hole. This score favors actions whose realized next state is closer to the goal and penalizes unsafe transitions. Because FrozenLake is stochastic, the score is applied to the observed post-transition state s_{t+1}, not to a deterministic prediction of the intended action outcome.

WebShop. WebShop is deterministic but exhibits delayed rewards since task success is only determined after the final purchase. Consequently, the environment reward provides little guidance for ranking intermediate navigation actions. We therefore use a lightweight state-based score based on task progress and instruction–page alignment. Let u denote the user instruction, h_t the interaction history, and p_t the textual content of the current webpage. After executing action a_t, we define

$$S_{\text{WS}}(h_t, a_t) = \alpha \cdot \mathbb{I}[\text{progress}(h_{t+1})] + \beta \cdot \text{sim}(u, p_{t+1}) \,, \tag{10}$$

where progress$(\cdot)$ is a binary indicator for task milestones, such as reaching a product page, selecting required options, or entering checkout. The term sim(u, p_{t+1}) is a low-cost textual similarity measure between the instruction and the resulting page content, implemented with simple keyword or attribute overlap. The weights $\alpha, \beta > 0$ balance structural progress and semantic alignment. This score is intentionally simple: it is not intended to replace the task reward, but only to rank candidate branches during rollout generation.

E Training and Evaluation Setup

Our training setup closely follows the experimental configuration of RAGEN (Wang et al., 2025). We summarize the key parameters and explicitly note deviations. Unless stated otherwise, all environment settings, optimization details, and implementation choices are identical to those used in the default RAGEN configurations.

Models and Optimization. We initialize all agents from Qwen2.5-Instruct (Qwen et al., 2025) checkpoints. For Sokoban and FrozenLake, we train both Qwen2.5-0.5B and Qwen2.5-3B models, while for WebShop we train only Qwen2.5-3B due to its longer context and higher reasoning demands. Policy optimization is done using PPO for Sokoban and GRPO for FrozenLake and WebShop, consistent with Wang et al. (2025)'s findings on stability under stochastic and sparse-reward settings.

Rollout and Update Configuration. Training proceeds for up to 200 rollout–update iterations for Sokoban and FrozenLake, and 100 iterations for WebShop. At each iteration, we sample $P = 16$ task instances (prompts) and retain $L_{\text{train}} = 16$ trajectories per task group before group-level instance filtering. We distinguish these retained trajectories from additional candidates generated during rollout construction, denoted by L_{gen} when applicable. For standard instance-filtered rollout sampling, $L_{\text{gen}} = L_{\text{train}} = 16$. Each trajectory is limited to a maximum of 5 interaction turns for Sokoban and FrozenLake and 10 turns for WebShop. For TSR (Best-of-N), we sample $N = 32$ complete candidate trajectories and retain the top $L_{\text{train}} = 16$, corresponding to a nominal complete-trajectory sampling budget of $N/L_{\text{train}} = 2\times$ relative to standard instance-filtered rollout sampling. For TSR (Beam Search), we use a total per-turn expansion budget of $M = 4$ across $B = 2$ active beams. The M candidate expansions are not divided uniformly across the B active beams but allocated according to their accumulated search scores, so higher-scoring prefixes can receive several candidates while lower-scoring prefixes receive none. Consequently, even $M = B$ performs selection, and at $M = B = 2$ beam search operates at a $1\times$ rollout-generation budget. The $L_{\text{train}} = 16$ retained trajectories are obtained from $L_{\text{train}}/B = 8$ independent beam-search trees, each retaining $B = 2$ terminal trajectories. Beam search therefore executes approximately $(L_{\text{train}}/B)MK$ candidate transitions per task group, compared with $L_{\text{train}}K$ for standard instance-filtered rollout sampling,

yielding a nominal relative rollout-generation budget of $M/B = 2\times$. For TSR (Lookahead), we run $L_{\text{train}} = 16$ independent searches, each producing one retained trajectory. At every realized decision, we evaluate $M = 4$ candidate actions and follow each candidate with one continuation of up to $D = 2$ additional environment transitions. The resulting nominal transition count per task group is

$$T_{\text{LA}} = L_{\text{train}} M \sum_{t=0}^{K-1} \left[1 + \min(D, K - 1 - t)\right],$$

corresponding to approximately $9.6\times$ the naive transition count for Sokoban and FrozenLake ($K = 5$) and $10.8\times$ for WebShop ($K = 10$), before accounting for early termination. To keep this tractable, candidate actions and continuations are generated without reasoning traces, while the retained trajectory keeps the lightweight outputs. Consequently, although lookahead executes substantially more environment transitions, its measured compute is approximately 1.7–$2.1\times$ that of instance-filtered sampling per update step (see Table 12). Each root candidate is ranked by the undiscounted cumulative score along its continuation. Throughout our compute accounting, all candidate generations and environment transitions used during search are counted, including branches that are evaluated but not retained for policy optimization.

Search and Scoring Hyperparameters. For FrozenLake, we set the hole-avoidance penalty in Eq. 9 to $\lambda = 5.0$, such that entering a hole is penalized more strongly than any single-step reduction in Manhattan distance. For WebShop, we use $\alpha = 1.0$ and $\beta = 0.5$ in Eq. 10, giving greater weight to discrete task-progress milestones while using instruction–page similarity as a secondary ranking signal. For the Monte Carlo-backed search in App. F.7, each candidate prefix is scored by the mean terminal task reward of its Monte Carlo continuations, which incurs additional rollout-generation compute beyond the matched $2\times$ comparisons. For the noisy-score robustness experiments, we min–max normalize candidate scores to $[0, 1]$ within each decision step before adding $\epsilon \sim \mathcal{N}(0, \sigma^2)$. Accordingly, $\sigma = 0.25$ and $\sigma = 0.50$ correspond to noise standard deviations of 25% and 50% of the normalized score range, respectively.

Rollout-Budget-Matched Baselines. To assess whether TSR's gains stem from structured search rather than raw rollout budget, we additionally evaluate compute-scaled instance-filtering baselines across all tasks with Qwen2.5-3B. We keep the number of task groups $P = 16$ and the downstream rollout set $L_{\text{train}} = 16$ fixed, while increasing the number of naively generated complete trajectories per task group from $L_{\text{gen}} = 16$ to 32 and 64. These settings correspond to nominal rollout-generation budgets of $1\times$, $2\times$, and $4\times$, respectively, while leaving the downstream optimizer and evaluation protocol unchanged. This isolates the effect of structured search from simply allocating additional generation compute to independent complete trajectories. Actual wall-clock and total-compute costs are reported separately in App. F.

Optimization Hyperparameters. We use the Adam optimizer with learning rate 1×10^{-6} for both actor and critic (when applicable), and $(\beta_1, \beta_2) = (0.9, 0.999)$. For PPO, advantages are computed using Generalized Advantage Estimation (GAE) (Schulman et al., 2018) with $\gamma = 1.0$ and $\lambda = 1.0$, and a clipping parameter $\epsilon = 0.2$. For GRPO, advantages are computed via group-relative normalization over rollouts sampled from the same task instance. Entropy regularization with coefficient $\beta = 0.001$ is applied in all experiments. Following RAGEN, we impose a response-format penalty of -0.1 when the model fails to produce a valid structured output.

Batching and Infrastructure. The update batch size is set to $E = 32$, with a mini-batch size of 4 per GPU. All experiments are conducted on 8 NVIDIA A100 GPUs using Fully Sharded Data Parallel (FSDP) training. Rollout generation is accelerated using vLLM with retained computation graphs across prefill and sampling, and distributed execution is handled via Ray.

Evaluation Protocol. Evaluation is performed on a fixed set of 256 held-out prompts per environment. Decoding uses temperature $T = 0.5$, with episodes truncated after 5 turns for Sokoban and FrozenLake and 10 turns for WebShop. We report the following metrics:

1. **Success Rate.** Final Task Accuracy on the fixed validation prompts, e.g., solving the puzzle in Sokoban, reaching the goal in FrozenLake, and completing a successful purchase in WebShop.

2. **Rollout Entropy.** Average token-level entropy of sampled responses, used as an indicator for exploration and policy uncertainty. Sharp drops may indicate premature convergence or collapse.
3. **Average Response Length.** Mean number of generated tokens per rollout, capturing verbosity and, in our setup, the extent of reasoning traces.
4. **Gradient Norm.** ℓ_2 norm of the policy gradient vector as an indicator of training stability. Sudden spikes correlate with unstable updates and potential collapse.
5. **Average Number of Turns.** Average number of interaction turns per episode before termination (success, failure, or truncation), used to measure whether TSR reduces interaction budget at inference time after training.

F Supplementary Results

In this section, we provide additional ablations and extended results.

F.1 Inference Efficiency

Complementing our inference cost analysis in Sec. 5.3, we provide an illustrative WebShop example showing how TSR can reduce redundant interactions at deployment time. In particular, the instance filtering baseline enters a looping or backtracking pattern, whereas TSR (Beam Search) follows a shorter, more goal-directed trajectory.

a) Instance Filtering: longer, looping behavior

Turn 1
Instruction: `Find decorative pillows ... color beige latte ... size 20”x20” ... under $30 ...`
Output: `<think>Search for beige pillows with size 20x20.</think> <answer>search[bisque pillow beige 20x20]</answer>`
Reward: `0.0`

Turn 2
State: Search results page (50 items).
Output: `<think>Click a result and then filter color/size.</think> <answer>click[b08l2zdwn2]</answer>`
Reward: `0.0`

Turn 3
State: Filters shown (`color: beige, black, ...,size: 18”x18”, ...`).
Output: `<think>Try clicking color filters.</think> <answer>click[beige latte]</answer>`
Reward: `0.0`

Turn 4–8
Output: `(repeated filter clicks / back-and-forth navigation)`
Reward: `0.0`
(*Agent exhausts actions or remains in an unproductive loop.*)

b) TSR (Beam Search): shorter, goal-directed behavior

Turn 1
Output: `<think>Use a targeted query to reduce backtracking.</think> <answer>search[decorative pillow cover beige latte]</answer>`
Reward: `0.0`

Turn 2
State: Search results page.
Output: `<think>Select the most compatible result, then finish checkout.</think> <answer>click[b09f6scw47]</answer>`
Reward: `0.0`

Turn 3
State: Product page with options + `buy now`.
Output: `<answer>click[buy now]</answer>`
Reward: `+1.0`

Similar qualitative patterns can be observed in other WebShop instances, with TSR exhibiting less redundant navigation and shorter interaction trajectories.

Interpretation of Inference-Efficiency Metrics. Average response length and interaction turns are measured over complete evaluation episodes and therefore are not independent of task success, since successful episodes may terminate earlier. We consequently interpret these metrics as measures of the realized end-to-end inference cost of the learned policy, rather than as success-conditioned measures of efficiency. Thus, the observed reductions capture the combined effect of improved task completion and more concise interaction behavior at deployment time.

F.2 Rollout-Budget-Matched Baselines

Because TSR incurs additional compute during rollout generation, an important question is whether its gains can be matched by allocating comparable rollout-generation compute to naive stochastic sampling. To this end, we extend our analysis in Sec. 5.4 and construct compute-scaled instance-filtering baselines across Sokoban, FrozenLake, and WebShop with Qwen2.5-3B. For all comparisons, we keep the number of task groups $P = 16$, the number of trajectories retained for policy optimization $L_{\text{train}} = 16$, the downstream optimizer, training schedule, evaluation protocol, and model architecture fixed. For instance filtering, we increase the number of independently generated complete trajectories per task group from $L_{\text{gen}} = 16$ to 32 and 64, corresponding to nominal rollout-generation budgets of $1\times$, $2\times$, and $4\times$, respectively. TSR (Beam Search) uses $M = 4$ total candidate expansions per turn with $B = 2$ active beams, corresponding to a nominal rollout-generation budget of $M/B = 2\times$ relative to standard instance-filtered rollout sampling.

As shown in Table 7, increasing the naive rollout budget yields only modest improvements across all tasks. On Sokoban, increasing L_{gen} from 16 to 32 and 64 improves success from 43.7% to 44.8% and 45.2%, whereas TSR (Beam Search) reaches 52.3% under the matched $2\times$ rollout-generation budget. On FrozenLake, the same scaling improves success only from 48.7% to 49.5% and 49.8%, while TSR reaches 60.7% at $2\times$. On WebShop, increasing the naive rollout budget improves success from 70.3% to 72.1% and 72.8%, while TSR reaches 85.3% at $2\times$. These results show that TSR's gains cannot be explained by additional rollout-generation compute alone. Instead, structured search allocates this compute to intermediate decision points, allowing the rollout generator to avoid locally poor prefixes and construct higher-signal trajectories. In contrast, naive scaling spends the additional compute on independently sampled complete trajectories and therefore cannot redirect rollout generation before irreversible or low-value decisions have already been made. Overall, under the same downstream training setup, structured per-turn search makes more effective use of a matched rollout-generation budget than simply increasing the number of naive stochastic rollouts.

Table 7: **Rollout-Budget-Matched Baselines across Tasks (Qwen2.5-3B).** All methods use $P = 16$ task groups and retain $L_{\text{train}} = 16$ trajectories per group for policy optimization. Scaling naive instance filtering yields limited gains compared with structured search under matched rollout-generation budgets.

Task	Method	Generation Config.	Rel. Rollout Budget	Success Rate
Sokoban	Instance Filtering	$L_{\text{gen}} = 16$	$1.0\times$	43.7 ± 1.6
	Instance Filtering	$L_{\text{gen}} = 32$	$2.0\times$	44.8 ± 1.7
	Instance Filtering	$L_{\text{gen}} = 64$	$4.0\times$	45.2 ± 1.6
	TSR (Beam Search)	$M = 4,\ B = 2$	$2.0\times$	$\mathbf{52.3 \pm 1.3}$
FrozenLake	Instance Filtering	$L_{\text{gen}} = 16$	$1.0\times$	48.7 ± 1.9
	Instance Filtering	$L_{\text{gen}} = 32$	$2.0\times$	49.5 ± 1.7
	Instance Filtering	$L_{\text{gen}} = 64$	$4.0\times$	49.8 ± 1.6
	TSR (Beam Search)	$M = 4,\ B = 2$	$2.0\times$	$\mathbf{60.7 \pm 1.2}$
WebShop	Instance Filtering	$L_{\text{gen}} = 16$	$1.0\times$	70.3 ± 1.8
	Instance Filtering	$L_{\text{gen}} = 32$	$2.0\times$	72.1 ± 1.6
	Instance Filtering	$L_{\text{gen}} = 64$	$4.0\times$	72.8 ± 1.5
	TSR (Beam Search)	$M = 4,\ B = 2$	$2.0\times$	$\mathbf{85.3 \pm 1.1}$

F.3 Scaling Search Budgets

Complementing our analysis in Sec. 5.5, we investigate the effect of scaling the search-budget parameters of TSR (Beam Search) in Tables 8, 9, and 10 for Sokoban, FrozenLake, and WebShop, respectively. Across the three environments, increasing the beam width from $B = 1$ to $B = 2$ generally improves performance, particularly at moderate and larger action-sampling budgets. In contrast, increasing the number of candidate actions M exhibits diminishing returns, with performance gradually saturating at larger sampling budgets. Although $M = 6$ often achieves the highest mean success rate, its gains over $M = 4$ are small relative to the additional rollout-generation cost. We therefore use moderate search budgets (e.g., $B = 2$, $M = 4$) in the main experiments as a favorable performance–rollout-compute trade-off. Similar trends can be observed for TSR (Best-of-N) and TSR (Lookahead) variants, which we omit for brevity.

Table 8: **Sokoban: Beam Search Scaling.** Increasing beam width (B) from 1 to 2 yields the largest gain. Increasing samples (M) shows diminishing returns.

Budget	Qwen2.5-0.5B			Qwen2.5-3B		
	Success Rate (↑)	Resp. Len (↓)	Turns (↓)	Success Rate (↑)	Resp. Len (↓)	Turns (↓)
M=2, B=1	36.8 ± 1.5	101	3.95	50.6 ± 1.4	156	3.70
M=4, B=1	37.6 ± 1.4	99	3.88	51.4 ± 1.3	154	3.65
M=6, B=1	37.9 ± 1.4	98	3.85	51.8 ± 1.4	153	3.62
M=2, B=2	37.9 ± 1.1	96	3.75	53.1 ± 1.1	155	3.75
M=4, B=2	38.3 ± 1.3	98	3.80	52.3 ± 1.3	152	3.60
M=6, B=2	39.4 ± 1.2	95	3.72	53.4 ± 1.2	149	3.52

Table 9: **FrozenLake: Beam Search Scaling.** Performance trends are consistent with Sokoban, where wider beams ($B = 2$) outperform width-one search ($B = 1$) across all sample counts.

Budget	Qwen2.5-0.5B			Qwen2.5-3B		
	Success Rate (↑)	Resp. Len (↓)	Turns (↓)	Success Rate (↑)	Resp. Len (↓)	Turns (↓)
M=2, B=1	28.7 ± 1.5	160	3.65	58.9 ± 1.4	110	3.20
M=4, B=1	29.3 ± 1.4	157	3.58	59.6 ± 1.3	106	3.16
M=6, B=1	29.5 ± 1.4	156	3.56	59.9 ± 1.3	104	3.14
M=2, B=2	29.4 ± 1.2	157	3.66	59.5 ± 1.1	98	3.16
M=4, B=2	30.0 ± 1.4	152	3.50	60.7 ± 1.2	98	3.10
M=6, B=2	30.8 ± 1.3	149	3.46	61.5 ± 1.2	95	3.07

Table 10: **WebShop: Beam Search Scaling (Qwen2.5-3B).** Performance generally improves with wider beams at moderate and larger sampling budgets, while gains saturate as M increases.

Budget	Qwen2.5-3B		
	Success Rate (↑)	Resp. Len (↓)	Turns (↓)
M=2, B=1	83.6 ± 1.3	470	6.05
M=4, B=1	84.5 ± 1.2	462	5.90
M=6, B=1	84.8 ± 1.2	458	5.86
M=2, B=2	83.4 ± 1.2	461	5.93
M=4, B=2	85.3 ± 1.1	453	5.80
M=6, B=2	85.9 ± 1.1	446	5.72

F.4 Distribution Shift and KL Analysis

TSR induces a search-guided rollout distribution $\mu_{\theta,\phi}$ by sampling candidate actions from the current policy π_θ and selecting among them using the search strategy $\mathcal{F}_\phi$. As discussed in App. B.2, this distribution is generally not identical to naive on-policy sampling. We therefore quantify how strongly search changes the rollout distribution as the search budget increases.

We estimate distribution shift using the average per-step KL divergence between the action distribution induced by TSR selection and the policy that generated the candidate actions. At each search step, we restrict both distributions to the sampled candidate set. Here, q_π denotes the policy probabilities renormalized over these candidates, while q_{TSR} denotes the corresponding normalized selection distribution induced by search. We report $D_{\mathrm{KL}}(q_{\mathrm{TSR}} \,\|\, q_\pi)$ averaged over rollout steps and training batches, and use it as a local measure of search-induced selection shift rather than as an exact trajectory-level $D_{\mathrm{KL}}(\mu_{\theta,\phi} \,\|\, \pi_\theta)$.

In this sweep, the per-turn expansion budget scales with beam width as $M = 2B$. For beam widths that do not divide $L_{\mathrm{train}} = 16$, we run $\lceil L_{\mathrm{train}}/B \rceil$ independent beam-search trees, pool their terminal trajectories, and retain the top L_{train} trajectories for policy optimization, while excess terminal trajectories are discarded. This yields the realized rollout-transition budgets reported in Table 11. Accordingly, wider beams do not receive less rollout compute.

Complementing our insights from Sec. 5.5, the results in Table 11 show a clear search-budget boundary. For $B \leq 5$, the induced KL divergence remains small ($\sim$ 0.02–0.05), indicating that search mostly selects among actions already plausible under the current policy. In this regime, TSR improves rollout quality while remaining policy-proximal. At $B = 10$, the KL divergence increases to ~ 0.14, and the additional performance gain becomes marginal. For $B \geq 20$, the KL divergence exceeds 0.30 and performance degrades. Since these settings use equal or greater rollout compute than moderate beam widths, the degradation cannot be attributed to insufficient compute. It instead suggests that the selected trajectories are too far from the proposing policy for clipped PPO/GRPO-style updates to extract useful gradients.

This analysis clarifies the practical role of the search width in TSR (Beam Search). The goal is not to approximate an unconstrained oracle policy but to improve rollout quality while staying near the current policy. Moderate search therefore acts as a controlled policy-improvement operator, whereas excessively wide search can induce harmful distribution shift.

Table 11: **Distribution Shift Analysis (FrozenLake, Qwen2.5-3B).** The per-turn expansion budget scales with beam width ($M = 2B$), so wider beams never receive less rollout compute (2.0–3.75×). Moderate beam widths remain policy-proximal and improve learning, while wider search increases KL divergence and degrades performance.

Beam Width B	M	Rollout Budget	Avg. KL / Step	Success Rate	Observation
2	4	2.0×	~ 0.02	60.7 ± 1.2	Safe policy-proximal regime
5	10	2.5×	~ 0.05	$\mathbf{61.3 \pm 1.1}$	Small additional gain
10	20	2.5×	~ 0.14	60.9 ± 1.4	Diminishing returns, KL rising
20	40	2.5×	~ 0.31	53.8 ± 1.8	Performance degrades
30	60	3.75×	~ 0.48	45.1 ± 2.1	Learning becomes unstable

F.5 Convergence and Compute Analysis

Shifting trajectory search from inference time to training time increases the rollout generation cost of individual updates. However, the relevant question is not only the cost of a single update but also how quickly each method reaches a strong policy. We therefore analyze both convergence in optimization steps and total training cost, reporting wall-clock time and estimated FLOPs to convergence.

Table 12 reports convergence statistics for Sokoban, FrozenLake, and WebShop. Across all tasks and model sizes, TSR variants generally converge in fewer optimization steps than instance filtering while reaching higher final success rates. The reduction is most pronounced for TSR (Beam Search), which requires up to 50% fewer convergence steps on Sokoban and over 40% fewer steps on FrozenLake and WebShop. Thus, TSR trades more expensive rollout-generation steps for fewer policy updates and higher final performance.

Importantly, fewer optimization steps do not imply uniformly lower total training cost. For Qwen2.5-3B with TSR (Beam Search), wall-clock time and total compute to convergence remain close to the instance-filtering baseline on Sokoban (2.05 vs. 1.99 h; 3.68 vs. 3.58 EFLOPs) and are lower on FrozenLake (1.45 vs. 1.57 h; 2.61 vs. 2.82 EFLOPs). On WebShop, TSR incurs higher total training cost (7.47 vs. 5.31 h; 13.42 vs. 9.54 EFLOPs), while simultaneously yielding the largest improvement in final success rate (85.3% vs. 70.3%). We therefore view TSR as a performance–training-compute

trade-off rather than a universally compute-saving method. Together with the rollout-budget-matched results in App. F.2, these results show that TSR's structured search provides a more effective allocation of additional training-time rollout compute than simply scaling naive rollout sampling

Table 12: **Convergence and Compute Analysis Across Tasks.** We report final success rate (mean $\pm$ standard deviation across runs), steps to reach 80% of final performance, steps to convergence, the relative reduction in convergence steps compared to instance filtering, wall-clock time to convergence, and estimated total compute (as EFLOPs) to convergence.

Model / Task	Method	Success Rate	Steps to 80%	Steps to Conv.	Step Red.	Time to Conv.	EFLOPs to Conv.
Qwen2.5-0.5B (Sokoban)	Instance Filtering	29.0 ± 1.7	90	180	–	1.54 h	2.77
	TSR (Best-of-N)	33.3 ± 1.5	60	120	33.3%	**1.51 h**	**2.71**
	TSR (Lookahead)	36.1 ± 1.4	55	105	41.7%	1.63 h	2.93
	TSR (Beam Search)	$\mathbf{38.3 \pm 1.3}$	**50**	**90**	**50.0%**	1.66 h	2.98
Qwen2.5-3B (Sokoban)	Instance Filtering	43.7 ± 1.6	100	180	–	**1.99 h**	**3.58**
	TSR (Best-of-N)	47.7 ± 1.5	70	150	16.7%	2.53 h	4.55
	TSR (Lookahead)	49.5 ± 1.4	**45**	150	16.7%	2.96 h	5.32
	TSR (Beam Search)	$\mathbf{52.3 \pm 1.3}$	55	**90**	**50.0%**	2.05 h	3.68
Qwen2.5-0.5B (FrozenLake)	Instance Filtering	19.7 ± 1.8	80	120	–	**0.80 h**	**1.44**
	TSR (Best-of-N)	25.0 ± 1.6	70	100	16.7%	1.32 h	2.37
	TSR (Lookahead)	27.8 ± 1.5	**60**	90	25.0%	1.14 h	2.05
	TSR (Beam Search)	$\mathbf{30.0 \pm 1.4}$	**60**	**80**	**33.3%**	0.95 h	1.71
Qwen2.5-3B (FrozenLake)	Instance Filtering	48.7 ± 1.9	150	170	–	1.57 h	2.82
	TSR (Best-of-N)	51.0 ± 1.6	110	140	17.6%	2.43 h	4.37
	TSR (Lookahead)	57.0 ± 1.4	90	130	23.5%	2.08 h	3.74
	TSR (Beam Search)	$\mathbf{60.7 \pm 1.2}$	**80**	**100**	**41.2%**	**1.45 h**	**2.61**
Qwen2.5-3B (WebShop)	Instance Filtering	70.3 ± 1.8	78	85	–	**5.31 h**	**9.54**
	TSR (Best-of-N)	73.3 ± 1.5	60	70	17.6%	6.83 h	12.27
	TSR (Lookahead)	82.3 ± 1.3	**45**	65	23.5%	8.31 h	14.93
	TSR (Beam Search)	$\mathbf{85.3 \pm 1.1}$	**45**	**50**	**41.2%**	7.47 h	13.42

F.6 Disentangling Search from Scoring Signals

The main FrozenLake and WebShop experiments use state-/progress-based scoring signals to rank candidate actions during rollout construction. Because these signals provide additional training-time information beyond the sparse task reward, improvements over naive rollout sampling could in principle arise either from the scoring information itself or from how TSR exploits that information through structured search. We therefore perform a series of controlled experiments that progressively separate these effects. Unless stated otherwise, all experiments use Qwen2.5-3B and keep the task, policy optimizer, evaluation protocol, and policy-optimization reward unchanged.

Does search work using only the task reward? We first remove the state-/progress-based scorer entirely and guide rollout selection using only the original task reward. For Best-of-N, this corresponds to ranking completed trajectories by their terminal return. For beam and lookahead search, candidate prefixes are ranked using the task reward observed along the partial trajectory. Because FrozenLake and WebShop provide sparse or delayed rewards, most incomplete prefixes receive identical zero reward, making immediate task reward a weak signal for distinguishing intermediate branches.

Table 13 reports the results. All reward-only rollout-selection variants improve clearly over naive instance filtering, demonstrating that the state-/progress-based scoring signal is not required for TSR-style rollout improvement. At the same time, sparse immediate rewards provide little information for distinguishing different intermediate prefixes. Accordingly, the three search variants remain relatively close under raw task rewards. On FrozenLake, Best-of-N reaches 51.0%, beam search reaches 53.4%, and lookahead reaches 52.7%. On WebShop, the corresponding results are 73.3%, 75.9%, and 75.4%. These results suggest that the effectiveness of intermediate search depends in part on how well partial trajectories can be distinguished. The state-/progress-based score provides a more informative ranking signal and yields stronger performance. In App. F.7, we remove the need for any auxiliary prefix score by propagating the original terminal task reward to intermediate prefixes through Monte Carlo continuations.

Table 13: **Raw-Reward and Guided Rollout Selection (Qwen2.5-3B).** We compare rollout search using only the original task reward with the state-/progress-based scoring signals used in the main experiments. Success rates are reported as mean ± standard deviation across runs.

Method	Rollout Scorer	FrozenLake	WebShop
Instance Filtering	None	48.7 ± 1.9	70.3 ± 1.8
TSR (Best-of-N)	Raw task reward	51.0 ± 1.6	73.3 ± 1.5
TSR (Lookahead)	Raw task reward	52.7 ± 1.7	75.4 ± 1.6
TSR (Beam Search)	Raw task reward	53.4 ± 1.5	75.9 ± 1.4
TSR (Lookahead)	State-/progress-based	57.0 ± 1.4	82.3 ± 1.3
TSR (Beam Search)	State-/progress-based	$\mathbf{60.7 \pm 1.2}$	$\mathbf{85.3 \pm 1.1}$

Does access to the scoring signal explain the gains? We next directly control for the information available to the rollout generator. We construct information-matched baselines in which the same state-/progress-based score is available to all score-guided methods, while varying how that information is used during trajectory construction. *Score-Augmented Instance Filtering* independently samples complete trajectories and uses the score only for trajectory selection, without intermediate branching or pruning. *Greedy Score-Guided Search* applies the same score during rollout construction but retains only the highest-scoring prefix at each step. TSR Beam Search instead maintains multiple promising prefixes and repeatedly reallocates rollout compute among them.

Table 14 shows that access to the scoring signal alone does not reproduce TSR's performance. Providing the same score to instance filtering improves success to 55.6% on FrozenLake and 79.2% on WebShop. Greedy score-guided search further improves performance to 56.9% and 80.6%. TSR Beam Search reaches 60.7% and 85.3%, respectively. Thus, under identical scoring information, TSR improves over score-augmented instance filtering by 5.1 percentage points on FrozenLake and 6.1 points on WebShop. It also improves over greedy per-turn selection by 3.8 and 4.7 points. These results show that the benefit of TSR is not explained by access to the scoring signal itself. Structured trajectory construction provides a substantial additional gain by using the same information to preserve competing prefixes, avoid premature commitment, and redirect rollout compute toward more promising regions of the trajectory space. In App. F.7, we further examine structured search when prefixes are ranked using only the original terminal task reward.

Table 14: **Information-Matched Scoring Controls (Qwen2.5-3B).** All score-guided methods use the same state-/progress-based scoring signal, while differing in how it is used during rollout construction.

Method	Aux. Scorer	Per-Turn Search	FrozenLake	WebShop
Instance Filtering	No	No	48.7 ± 1.9	70.3 ± 1.8
Score-Augmented Instance Filtering	Yes	No	55.6 ± 1.6	79.2 ± 1.5
Greedy Score-Guided	Yes	Greedy	56.9 ± 1.4	80.6 ± 1.3
TSR (Beam Search)	Yes	Beam	$\mathbf{60.7 \pm 1.2}$	$\mathbf{85.3 \pm 1.1}$

How sensitive is TSR to the scoring signal? Finally, we test whether TSR depends on the particular state-/progress-based scoring formulation used in the main experiments. We compare the original task reward with a task-agnostic policy-confidence score based on negative action entropy, a learned observation-only value estimate, and noisy versions of the state-/progress-based score. For the noisy-score experiments, we add zero-mean Gaussian noise with increasing standard deviation. The learned value model uses only information available from the agent's observation stream. For each environment, the value model is trained on baseline-generated trajectories using only the observations and interaction history available to the agent to predict eventual terminal success.

Table 15 shows that TSR remains effective across qualitatively different scoring mechanisms. Task-agnostic entropy improves over raw-reward search on both tasks, showing that TSR can exploit a generic policy-derived ranking signal without task-specific progress features. The learned observation-only value model provides a further improvement and reaches 57.6% on FrozenLake and 81.4% on WebShop. TSR also remains robust when substantial noise is injected into the state-/progress-based score. Performance degrades gradually rather than collapsing as the ranking signal becomes less accurate. The strongest results are obtained with the state-/progress-based score, confirming that

more informative prefix evaluation further strengthens structured search. Crucially, however, TSR remains well above naive rollout generation across task-reward, task-agnostic, learned, and noisy scoring variants. The scoring function should therefore be viewed as a modular interface for ranking candidate prefixes rather than as a dependency on any particular form of intermediate feedback.

Table 15: **Robustness to Rollout Scoring Signals (Qwen2.5-3B).** We evaluate TSR (Beam Search) with task-reward, task-agnostic, learned, noisy, and state-/progress-based scoring signals.

Rollout Scorer	FrozenLake	WebShop
Instance Filtering (no scorer)	48.7 ± 1.9	70.3 ± 1.8
Raw task reward	53.4 ± 1.5	75.9 ± 1.4
Task-agnostic entropy score	55.8 ± 1.5	78.6 ± 1.4
Learned observation-only value	57.6 ± 1.3	81.4 ± 1.3
Noisy state-/progress score ($\sigma = 0.50$)	55.5 ± 1.6	80.5 ± 1.5
Noisy state-/progress score ($\sigma = 0.25$)	56.2 ± 1.4	80.8 ± 1.2
State-/progress-based score	$\mathbf{60.7 \pm 1.2}$	$\mathbf{85.3 \pm 1.1}$

Summary. Overall, these controls establish three complementary findings. First, the state-/progress-based score is not necessary for TSR-style rollout improvement, since reward-only variants already provide clear gains over naive instance filtering on both sparse-reward environments. Second, the auxiliary scoring information cannot by itself explain TSR's gains. When the scoring signal is held fixed, structured beam search delivers substantially larger improvements than both score-augmented trajectory selection and greedy per-turn selection. Third, TSR remains effective across task-reward, task-agnostic, learned, and noisy scoring functions, demonstrating that its performance is not tied to a specific rollout scorer. The remaining challenge under sparse rewards is thus not dependence on auxiliary scoring, but the lack of a discriminative estimate for intermediate prefixes. The following subsection addresses this issue by using Monte Carlo-backed terminal rewards to compare structured tree search against flat Best-of-N under matched scoring information.

F.7 Heuristic-Free Monte Carlo-Backed Search

The previous experiments show that sparse immediate task rewards provide limited information for ranking intermediate trajectory prefixes. We therefore ask a stricter question: *Can structured search exploit only the original terminal task reward when that outcome is propagated to intermediate prefixes through Monte Carlo continuations?* To answer this, we introduce a Monte Carlo-backed variant of TSR that propagates terminal task outcomes to intermediate prefixes without using state-/progress-based scores, task-agnostic scores, or learned value models.

At each intermediate prefix, we sample candidate actions from the current policy and generate Monte Carlo continuations to episode termination. Each candidate prefix is scored by the mean original terminal task reward obtained by its continuations. Formally, for a candidate prefix $\tau_{<t} \circ a_t$ with C sampled continuations, we estimate

$$\hat{S}_{\mathrm{MC}}(\tau_{<t}, a_t) = \frac{1}{C} \sum_{c=1}^{C} R\Big(\tau^{(c)}\Big), \tag{11}$$

where $R(\tau^{(c)})$ is the original terminal task reward of continuation c. Thus, the search receives no additional supervision beyond the task objective itself. All candidate actions and Monte Carlo continuations are included in the rollout-generation cost. Because these continuations introduce additional environment interactions beyond the standard search variants, we report the Monte Carlo-backed results as a heuristic-free robustness analysis rather than as a strictly compute-matched comparison.

We compare Best-of-N, greedy, lookahead, and beam variants under the same Qwen2.5-3B model, optimizer, task distribution, and terminal task reward. Best-of-N independently samples $N = 32$ complete trajectories and retains the top $L_{\mathrm{train}} = 16$ only after termination, corresponding to a $2.0\times$ rollout-generation budget. Greedy search uses Monte Carlo-backed prefix scores but commits to the highest-scoring prefix at each decision point. TSR Lookahead evaluates short continuation trees before committing, while TSR Beam Search maintains multiple promising prefixes throughout

rollout construction. We additionally include instance filtering at its standard $1.0\times$ budget and scaled instance filtering at $2.0\times$ and $4.0\times$ budgets to contextualize the effect of additional naive sampling. Results are averaged across three seeds.

As shown in Table 16, scaling naive rollout sampling produces only small improvements, from 48.7% to 49.5% and 49.8% on FrozenLake and from 70.3% to 72.1% and 72.8% on WebShop at $2.0\times$ and $4.0\times$ budgets, respectively. Best-of-N reaches 51.0% and 73.3% at a $2.0\times$ budget. When the same terminal task reward is propagated to intermediate prefixes through Monte Carlo continuations, structured search improves substantially further. Monte Carlo-backed greedy search reaches 57.0% on FrozenLake and 79.2% on WebShop, lookahead reaches 58.0% and 80.0%, and Beam Search reaches 59.1% and 82.9%. Compared with Best-of-N, Monte Carlo-backed Beam Search improves by 8.1 and 9.6 percentage points, respectively. These results show that the benefit of structured intermediate search does not depend on auxiliary state-/progress-based scoring signals, although propagating terminal outcomes through Monte Carlo continuations requires additional rollout-generation compute.

Table 16: **Monte Carlo-Backed Search with Terminal Task Rewards (Qwen2.5-3B).** We compare heuristic-free search methods that use only the original terminal task reward. Best-of-N and scaled instance filtering use the indicated rollout-generation budgets, while Monte Carlo-backed variants incur additional rollout-generation compute due to terminal continuations. Success rates are mean $\pm$ standard deviation across three seeds.

Method	Prefix Scoring	Rollout Cost	FrozenLake	WebShop
Instance Filtering	Terminal task reward	$1.0\times$	48.7 ± 1.9	70.3 ± 1.8
Scaled Instance Filtering	Terminal task reward	$2.0\times$	49.5 ± 1.7	72.1 ± 1.6
Scaled Instance Filtering	Terminal task reward	$4.0\times$	49.8 ± 1.6	72.8 ± 1.5
TSR (Best-of-N)	Terminal task reward	$2.0\times$	51.0 ± 1.6	73.3 ± 1.5
TSR (Greedy Search)	MC-backed terminal reward	Additional MC compute	57.0 ± 1.3	79.2 ± 1.2
TSR (Lookahead)	MC-backed terminal reward	Additional MC compute	58.0 ± 1.2	80.0 ± 1.1
TSR (Beam Search)	MC-backed terminal reward	Additional MC compute	$\mathbf{59.1 \pm 1.0}$	$\mathbf{82.9 \pm 1.0}$

Paired Outcome Analysis. Aggregate success rates could in principle be driven by a small subset of favorable task instances. We therefore perform a paired analysis comparing Monte Carlo-backed Beam Search and Best-of-N on the same task instances and random seeds. Pairs are formed at the $(\text{seed}, \text{evaluation instance})$ level, such that both methods are evaluated on the same held-out task under the corresponding trained seed. For FrozenLake, the terminal task return is binary, with $R = 1$ if the goal is reached and $R = 0$ otherwise. For WebShop, we analogously compare the terminal task outcome used for evaluation. A Beam Search win is recorded when its terminal return is higher than that of Best-of-N, a Best-of-N win when the reverse holds, and equal returns are counted as ties. As shown in Table 17, Beam Search obtains a higher return on 23.2% of FrozenLake pairs compared with 15.1% for Best-of-N, while the remaining 61.7% are ties. On WebShop, Beam Search wins 18.9% of pairs, Best-of-N wins 9.2%, and 71.9% are ties. The difference between the two non-tie rates matches the corresponding aggregate performance gaps in Table 16 up to rounding. Among pairs with different terminal outcomes, Beam Search wins 60.6% of FrozenLake comparisons and 67.3% of WebShop comparisons. The high tie rates are expected because both methods frequently succeed or fail on the same instances, while the asymmetric non-tie outcomes show that Monte Carlo-backed structured search more often succeeds on instances where the two methods differ.

Table 17: **Paired Beam Search vs. Best-of-N Outcome Analysis.** We compare evaluation outcomes of Monte Carlo-backed Beam Search and Best-of-N on matched $(\text{seed}, \text{evaluation instance})$ pairs. Both methods use only the original terminal task reward, while Beam Search incurs additional rollout-generation compute during training.

Outcome	FrozenLake	WebShop
TSR Beam achieves higher evaluation return	$\mathbf{23.2\%}$	$\mathbf{18.9\%}$
Best-of-N achieves higher evaluation return	15.1%	9.2%
Tie	61.7%	71.9%

Summary. These results isolate the role of supervision in structured search. Monte Carlo-backed Beam Search outperforms naive sampling and flat Best-of-N while using only the original terminal task reward. The improvement therefore does not depend on a state-/progress-based scoring signal. Instead, terminal task outcomes can provide an effective prefix-ranking signal when they are propagated through Monte Carlo continuations. This heuristic-free variant incurs additional rollout-generation compute, while the raw-reward comparison in Table 13 separately shows that per-turn Beam Search already outperforms Best-of-N under the standard $2.0\times$ rollout budget.

F.8 Comparison to Related Multi-Turn RL Baselines

We further compare TSR against recent methods that improve multi-turn RL through alternative rollout-generation or credit-assignment mechanisms. In particular, we consider ARPO (Dong et al., 2025a), which adaptively branches rollouts at high-uncertainty interaction points, and GiGPO (Feng et al., 2025), which introduces finer-grained group-based credit assignment around repeated intermediate states. These methods are closely related in motivation but intervene at different stages of the training pipeline. ARPO determines where additional rollout exploration should occur based on policy uncertainty, while GiGPO refines credit assignment across intermediate states. TSR instead modifies the rollout generator itself by maintaining, scoring, and pruning alternative environment-level trajectory prefixes before policy optimization. We therefore evaluate whether structured trajectory search provides additional benefits beyond adaptive branching and improved credit assignment.

Baseline Implementation Details. We implement ARPO and GiGPO within RAGEN such that the model initialization, optimizer, learning rate, clipping parameters, task distribution, rollout budget, and evaluation protocol remain identical across methods, ensuring a fair comparison. Because ARPO was originally formulated around high-uncertainty tool interactions, we adapt its entropy-triggered branching to environment interaction steps. Specifically, we rank interaction steps within each rollout group by token-averaged action entropy and branch at the top 25% highest-entropy steps, sampling 4 candidate actions at each branch point and retaining all resulting continuations subject to the same global $2\times$ rollout-generation budget used for TSR. We retain ARPO's soft step-wise advantage attribution for trajectories sharing a common prefix. For the information-matched comparison in Table 19, the entropy-based setting corresponds to standard ARPO, whose branching is already driven by action entropy. For the state-/progress-based setting, ARPO retains its branching rule and advantage attribution, but when the $2\times$ budget does not permit keeping all continuations, the state-/progress-based signal determines which continuations are retained. This adds a score-based selection step absent from the original ARPO, so we denote this variant as ARPO + Score Selection. For GiGPO, anchor states are identified by exact matches of the environment observation, step-level groups contain up to 4 trajectories and require at least 2 members, and trajectory-level and step-level relative advantages are combined as $A = (1 - \lambda_{\text{step}})A_{\text{traj}} + \lambda_{\text{step}}A_{\text{step}}$ with $\lambda_{\text{step}} = 0.5$. Hyperparameters for both baselines are selected using a validation sweep of comparable size to the TSR search-budget ablations in App. F.3 and then fixed across tasks. Under this matched setup, both methods reproduce the qualitative improvements over instance filtering reported in the original works, although absolute results are not directly comparable because of differences in model backbone, rollout budget, and environment configuration.

Comparison under the Standard Training Setup. We first compare all methods under matched model initialization, task distribution, optimizer, rollout generation budget, policy-optimization reward, and evaluation protocol. Table 18 reports results on WebShop, which provides a representative long-horizon tool-use setting. Both ARPO and GiGPO improve clearly over instance filtering, confirming that stronger rollout generation and credit assignment are effective for multi-turn RL. TSR nevertheless achieves the highest final success rate, reaching 85.3% compared with 75.2% for ARPO and 80.2% for GiGPO. TSR further produces shorter responses, requires fewer interaction turns, and reaches convergence in fewer rollout–update steps. Thus, TSR maintains a clear performance advantage over both ARPO and GiGPO under the matched training setup.

Information-Matched Comparison with ARPO. A direct comparison between rollout methods can otherwise be influenced by the information available to the respective rollout-selection mechanisms. We therefore additionally compare ARPO and TSR under identical rollout-scoring information and matched rollout-generation budgets. We consider both a task-agnostic entropy signal and the

Table 18: **Comparison to Related Multi-Turn RL Baselines (WebShop, Qwen2.5-3B).** All methods use matched model initialization, task distribution, optimizer, rollout-generation budget, policy-optimization reward, and evaluation protocol. Success rates are reported as mean ± standard deviation across three runs.

Method	Success Rate (↑)	Resp. Len (↓)	Turns (↓)	Steps to Conv. (↓)
Instance Filtering	70.3 ± 1.8	519	6.8	85
ARPO	75.2 ± 1.5	492	6.3	63
GiGPO	80.2 ± 1.3	478	6.1	59
TSR (Beam Search)	$\mathbf{85.3 \pm 1.1}$	**453**	**5.8**	**50**

Table 19: **Information-Matched Comparison with ARPO (WebShop, Qwen2.5-3B).** ARPO and TSR use identical rollout-scoring information and matched rollout-generation budgets.

Method	Rollout Scorer	Success Rate	Steps to Conv.
ARPO	Task-agnostic entropy	75.2 ± 1.5	63
TSR (Beam Search)	Task-agnostic entropy	$\mathbf{78.6 \pm 1.4}$	**56**
ARPO + Score Selection	State-/progress-based	80.4 ± 1.3	58
TSR (Beam Search)	State-/progress-based	$\mathbf{85.3 \pm 1.1}$	**50**

state-/progress-based signal used in our main experiments. This comparison isolates how effectively each rollout mechanism uses the same intermediate information. As shown in Table 19, TSR outperforms ARPO under both scoring conditions. Using the task-agnostic entropy score, TSR reaches 78.6% compared with 75.2% for ARPO. When both methods receive the same state-/progress-based score, TSR reaches 85.3% compared with 80.4% for ARPO, corresponding to a 4.9 percentage-point improvement. TSR also reaches convergence in fewer rollout–update steps under both settings. These results show that TSR's advantage over ARPO is not explained by access to a different scoring signal. Instead, structured maintenance and pruning of concrete trajectory prefixes enables TSR to make more effective use of the same intermediate information.

Complementarity with Improved Credit Assignment. GiGPO and TSR intervene at different stages of the learning pipeline. GiGPO refines the learning signal assigned to intermediate decisions, while TSR modifies which trajectories are generated and retained before the policy update. The two mechanisms can therefore be combined directly. To test this, we apply TSR rollout generation together with GiGPO while keeping the remaining training configuration unchanged. Table 20 shows that combining the two methods yields further improvements over GiGPO alone. On FrozenLake, success increases from 56.8% with GiGPO to 62.4% with GiGPO+TSR. On WebShop, performance increases from 80.2% to 87.0%. The combined method also exceeds TSR alone on both tasks, providing direct evidence that trajectory search and finer-grained credit assignment address complementary aspects of multi-turn RL. These results support the modular interpretation of TSR as a rollout-generation mechanism that can be composed with improvements to the downstream policy-update procedure.

Table 20: **Complementarity of TSR and GiGPO (Qwen2.5-3B).** TSR modifies rollout construction, while GiGPO provides finer-grained credit assignment. Success rates are reported as mean ± standard deviation across runs.

Method	FrozenLake	WebShop
GiGPO	56.8 ± 1.5	80.2 ± 1.3
TSR (Beam Search)	60.7 ± 1.2	85.3 ± 1.1
GiGPO + TSR	$\mathbf{62.4 \pm 1.2}$	$\mathbf{87.0 \pm 1.0}$

Summary. These comparisons show that TSR remains effective against strong multi-turn RL baselines rather than only against naive rollout sampling. TSR achieves higher performance than both ARPO and GiGPO under the matched training setup. Its advantage over ARPO persists when both methods receive identical rollout-scoring information, directly separating the contribution of the rollout-search mechanism from that of the scoring signal. Moreover, combining TSR with GiGPO

yields further improvements over either method alone, showing that structured trajectory search is complementary to finer-grained credit assignment. Overall, these results establish TSR as a distinct and composable mechanism for improving rollout generation in multi-turn RL.

F.9 Optimizer Compatibility

TSR modifies only the rollout-generation procedure and does not introduce an optimizer-specific loss, advantage estimator, or policy-update rule. We therefore describe TSR as *optimizer-compatible by design*. To verify that its empirical gains are not specific to the optimizer choices used in our main experiments, we perform a controlled comparison on Sokoban with Qwen2.5-3B using both PPO and GRPO. For each optimizer, we keep the model, task, rollout configuration, scoring function, and evaluation protocol fixed, and compare instance filtering against TSR (Beam Search). As shown in Table 21, TSR improves performance under both optimization methods. With PPO, TSR increases success from 43.7% to 52.3%, corresponding to an 8.6 pp improvement. With GRPO, success increases from 42.4% to 50.1%, corresponding to a 7.7 pp improvement. TSR also reduces average response length and the number of interaction turns under both optimizers. These consistent gains support the view that the benefit of search-guided rollout generation is not specific to either PPO or GRPO. Importantly, we do not claim that optimizer choice is irrelevant to absolute performance. Rather, TSR can be integrated with different policy-gradient optimizers without changing their respective update rules, and the rollout-search benefit persists across both optimizers considered here.

Table 21: **Optimizer Compatibility on Sokoban (Qwen2.5-3B).** We compare instance filtering and TSR (Beam Search) under PPO and GRPO while keeping the model, rollout, scoring, and evaluation configurations fixed. Success rates are reported as mean $\pm$ standard deviation across three runs.

Optimizer	Method	Success Rate (↑)	Resp. Len. (↓)	Turns (↓)
PPO	Instance Filtering	43.7 ± 1.6	161	4.1
	TSR (Beam Search)	$\mathbf{52.3 \pm 1.3}$	**152**	**3.6**
GRPO	Instance Filtering	42.4 ± 1.8	166	4.2
	TSR (Beam Search)	$\mathbf{50.1 \pm 1.5}$	**156**	**3.7**

F.10 Comparison to Larger Zero-Shot Models

Consistent with multi-turn agent RL evaluations in prior works (Wang et al., 2025), we additionally compare smaller TSR-trained Qwen2.5-0.5B agents against larger zero-shot models to contextualize the magnitude of the task-specific RL gains. Importantly, this comparison is not intended as a compute- or training-matched evaluation as the larger models are used without task-specific RL, whereas the agents are trained on the target environments. Instead, the goal is to illustrate that improving rollout generation during multi-turn RL can substantially close the performance gap to much larger generalist models, and in some cases exceed their zero-shot performance on specialized agentic tasks. As shown in Table 22, task-specific RL substantially improves the smaller Qwen2.5-0.5B agent model performance. With TSR (Beam Search), the 0.5B model reaches 38.30% success on Sokoban and 30.00% on FrozenLake, exceeding the zero-shot GPT-4o and Qwen2.5-72B baselines in this evaluation. We emphasize that this does not imply that the smaller TSR-trained model is generally stronger than these larger models. Rather, the result highlights that multi-turn RL with improved rollout generation can produce strong specialized agents, even from comparatively small backbones. This supports our broader conclusion that train-time trajectory search is an effective mechanism for converting additional training compute into improved agentic task performance.

F.11 Additional Agentic Environments

To evaluate whether the benefits of TSR extend beyond the three environments considered in the main experiments, we additionally evaluate TSR on ALFWorld (Shridhar et al., 2021) and AppWorld (Trivedi et al., 2024). These environments broaden the evaluation to two complementary agentic settings. ALFWorld evaluates long-horizon household interaction and planning through text-based environments aligned with embodied tasks, while AppWorld evaluates interactive tool use and coding across multiple simulated applications and APIs. We evaluate both Qwen2.5-3B-Instruct

Table 22: **Comparison to Larger Zero-Shot Models.** This comparison contextualizes the magnitude of task-specific RL gains and is not intended as a compute- or training-matched comparison.

Model / Method	Sokoban	FrozenLake
GPT-4o (zero-shot)	27.73	26.56
Qwen2.5-72B (zero-shot)	19.53	23.83
Qwen2.5-0.5B (Instance Filtering)	29.00	19.70
Qwen2.5-0.5B (TSR Best-of-N)	33.30	25.00
Qwen2.5-0.5B (TSR Lookahead)	36.10	27.80
Qwen2.5-0.5B (TSR Beam Search)	**38.30**	**30.00**

Table 23: **Additional Agentic Environments.** Results on ALFWorld and AppWorld for Qwen2.5-3B and Qwen2.5-7B. We report ALFWorld task success rate and AppWorld Task Goal Completion (TGC). Results are mean $\pm$ standard deviation across three runs.

Model	Method	ALFWorld SR (↑)	AppWorld TGC (↑)
Qwen2.5-3B	Instance Filtering	69.8 ± 1.8	20.4 ± 1.6
	ARPO	74.6 ± 1.6	23.6 ± 1.5
	GiGPO	78.3 ± 1.4	25.2 ± 1.4
	TSR (Beam Search)	$\mathbf{84.7 \pm 1.3}$	$\mathbf{31.5 \pm 1.2}$
Qwen2.5-7B	Instance Filtering	79.4 ± 1.5	28.7 ± 1.5
	ARPO	84.1 ± 1.4	32.1 ± 1.4
	GiGPO	89.3 ± 1.2	34.8 ± 1.3
	TSR (Beam Search)	$\mathbf{92.1 \pm 1.0}$	$\mathbf{40.6 \pm 1.1}$

and the larger Qwen2.5-7B-Instruct model to additionally test whether the benefits of trajectory search persist at a larger model scale. For each environment and model size, we follow the original benchmark task definitions and evaluation protocols, while all compared methods use the same task distribution, observation and action interfaces, policy optimizer, policy-optimization reward, and rollout budget. We compare TSR (Beam Search) against instance filtering, ARPO, and GiGPO. For ALFWorld, we train on the standard training games and report task success rate on the unseen evaluation split. For AppWorld, we train on the train split and evaluate on test-normal, reporting Task Goal Completion (TGC), which counts a task as completed only when all corresponding evaluation tests pass. In both environments, TSR ranks candidate prefixes using an observation-only value model trained analogously to App. F.6, i.e., on baseline-generated trajectories to predict eventual task completion from the agent's observations and interaction history. All scores are averaged across three independent runs. Table 23 summarizes the results.

Branching and Environment State. Branching requires restoring previously visited environment states. We checkpoint the full simulator state at each branch point, which for AppWorld includes the application databases and the agent's interpreter state. The same snapshot-restore interface is used for all methods that branch, and all transitions generated during branching count toward the matched rollout-generation budget. TSR therefore receives no additional task information but requires a checkpointable training environment, which we treat as an explicit applicability condition.

Performance. Across both environments and model scales, TSR achieves the highest mean task performance. For Qwen2.5-3B, TSR reaches 84.7% success on ALFWorld compared with 69.8% for instance filtering, 74.6% for ARPO, and 78.3% for GiGPO. On AppWorld, TSR achieves 31.5% TGC, compared with 20.4% for instance filtering, 23.6% for ARPO, and 25.2% for GiGPO. This corresponds to gains of 14.9 percentage points on ALFWorld and 11.1 points on AppWorld over instance filtering. The same trend persists with Qwen2.5-7B. TSR achieves 92.1% success on ALFWorld and 40.6% TGC on AppWorld, compared with 79.4% and 28.7% for instance filtering. TSR also outperforms both ARPO and GiGPO at the larger model scale, reaching the strongest performance across both additional environments. The corresponding improvements over instance filtering are 12.7 percentage points on ALFWorld and 11.9 points on AppWorld. These results show that the gains from search-guided rollout generation extend beyond the original planning, navigation, and web-interaction environments.

Model-Scale Robustness. Increasing the backbone size improves absolute performance for all methods without diminishing the benefit of structured rollout search. For TSR, scaling from Qwen2.5-3B to Qwen2.5-7B improves ALFWorld success from 84.7% to 92.1% and AppWorld TGC from 31.5% to 40.6%. Importantly, TSR maintains gains of approximately 11–15 percentage points over instance filtering across both environments and model scales. Its advantage over ARPO and GiGPO also persists at 7B, providing evidence that the benefits of trajectory-search rollouts are not confined to the smaller backbone used in the main experiments.

Summary. The additional evaluations extend the empirical scope of TSR along two complementary dimensions. First, TSR improves performance on household interaction and multi-application tool-use tasks that differ substantially from Sokoban, FrozenLake, and WebShop. Second, the improvement persists when scaling the policy from Qwen2.5-3B to Qwen2.5-7B. Together with the main experiments, these results provide evidence that structured train-time trajectory search is effective across a broader range of multi-turn agentic environments and model scales.

G Limitations and Broader Impact

Limitations. TSR introduces additional compute during rollout generation, although this cost is incurred only at training time and is partially offset by requiring fewer optimization steps to convergence. TSR also assumes that the environment can be branched, restored, or re-queried at intermediate states, which holds naturally for the digital and simulator-based environments common in LLM-agent research (e.g., web sandboxes, text games, and code interpreters), but may require learned environment models in non-resettable settings. Our focus is therefore on resettable or simulatable training environments, while extending TSR to less controllable real-world settings remains important future work. Furthermore, when immediate environment rewards are sparse, more informative prefix scoring can improve search efficiency. In our main experiments, we use simple state-/progress-based signals (e.g., Manhattan distance and keyword overlap), while less structured environments may instead benefit from learned value or process-reward models. Importantly, our controls show that TSR remains effective with task rewards, task-agnostic signals, learned observation-only values, noisy scores, and Monte Carlo-backed terminal rewards, demonstrating that it is not tied to a particular scoring mechanism. Finally, our main evaluation spans two model scales and three complementary environments, with additional experiments extending the study to ALFWorld and AppWorld and to a 7B model. Evaluating further model families and interaction settings remains an important direction for assessing the broader applicability of TSR.

Broader Impact. TSR changes only training-time rollout generation and does not introduce additional deployment-time components or actions. By improving rollout quality, it can produce stronger policies with fewer interaction turns at inference time without requiring search at deployment, although this may come at additional training-time compute. Additionally, by enabling comparatively smaller models to achieve stronger task-specific performance, TSR may also lower the computational barrier to multi-turn agent research. As with other methods that improve agent capabilities, deployments involving external tools, transactions, code execution, or user data should retain appropriate safeguards and access controls.

Contributions. This work presents a systematic study of search-guided rollout generation for multi-turn RL of LLM agents. We introduce TSR, a policy-proximal and optimizer-compatible framework that moves structured trajectory search to training-time rollouts and distills the resulting behavior into the policy. Across five agentic environments and model scales from Qwen2.5-0.5B to Qwen2.5-7B, TSR consistently improves task performance while requiring no search at inference time. Our evaluation further isolates the contribution of structured search through rollout-budget-matched baselines, information-matched scoring controls, Monte Carlo-backed search, optimizer comparisons, evaluations against related multi-turn RL baselines, and extensive analyses of convergence, compute, and distribution shift. Together, these results establish TSR as a modular and empirically validated mechanism for improving rollout generation in multi-turn agent learning.