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

Aladin Djuhera[1], **Swanand Ravindra Kadhe**[2], **Farhan Ahmed**[2], **Heiko Ludwig**[2], **Holger Boche**[1]
[1] Technical University Munich {aladin.djuhera,boche}@tum.de
[2] IBM Research {swanand.kadhe,farhan.ahmed,hludwig}@ibm.com

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 high-quality trajectories by selecting high-scoring actions at each turn using task-specific feedback. This improves rollout quality and stabilizes learning while leaving the underlying optimization objective unchanged, making TSR optimizer-agnostic. We instantiate TSR with best-of-N, beam, and shallow lookahead search, and pair it with PPO and GRPO, achieving up to 15% performance gains and more stable learning on Sokoban, FrozenLake, and WebShop tasks at a one-time increase in training compute. By moving search from inference time to the rollout stage of training, TSR provides a simple and general 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., 2025b). 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: 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: *If multi-turn RL is allowed a fixed but non-trivial amount of additional compute at the trajectory rollout stage, how much can we improve the performance and stability of training?* 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 to train-time search during RL can reliably

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.

We introduce **TSR** (**T**rajectory-**S**earch **R**ollouts), a train-time framework that repurposes test-time search ideas to construct higher-quality trajectories. For each prompt, TSR runs a lightweight tree-style search over action sequences using available task feedback such as per-turn rewards or other metrics, and then selects the highest-scoring trajectory (or a subset) to compute advantages. TSR optimizes the trajectory *per-turn* and thus 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-agnostic and directly compatible with other existing frameworks.

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: Sokoban (Weber et al., 2018), a deterministic logic puzzle, FrozenLake (Brockman et al., 2016), a popular stochastic RL game, and WebShop (Yao et al., 2023a), a long-horizon, agentic web navigation benchmark. Under a fixed, one-time increase in train-time compute, **TSR yields up to 15% absolute performance gains** and significantly smoother training dynamics.

In summary, our main contributions are:

- **Trajectory Search Framework.** 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*.
- **Optimizer-Agnostic and Modular Design.** TSR changes only *how* rollouts are generated, making it compatible with standard policy-gradient methods (e.g., PPO, GRPO).
- **Empirical Validation.** We demonstrate that TSR improves performance on deterministic, stochastic, and long-horizon tasks, enabling smaller models to surpass the performance of larger models.

2 Background and Problem Setup

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

$$\tau = (a_0, o_0, r_0, \ldots, a_K, o_K, r_K) \tag{1}$$

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 $J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)]$, by performing gradient ascent in the direction of increasing expected return, i.e., $\theta_{\text{new}} = \theta + \eta \nabla_\theta J(\theta)$, where η denotes the learning rate. In the multi-turn setting, the corresponding policy gradient is given by

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{K-1} \nabla_\theta \log \pi_\theta(a_t \mid \tau_{<t})\, \hat{A}_t\right], \tag{2}$$

where $\hat{A}_t$ is an advantage estimate, measuring how much better action a_t is compared to the policy's average behavior.

In general, directly computing policy gradients is challenging because the gradient estimator typically suffers from high variance. To mitigate this, modern policy optimization methods, such as PPO (Schulman et al., 2017) and GRPO (Shao et al., 2024), 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 the multi-turn setting in App. B.

In this work, we do not alter these algorithms but instead modify the *rollout generation*. Rather than sampling purely from the current policy π_θ, we optimize rollouts on a per-turn level using tree-search methods to improve the quality of the data used to estimate $\hat{A}_t$.

2.3 Why Rollout Quality and Diversity Drive Stability

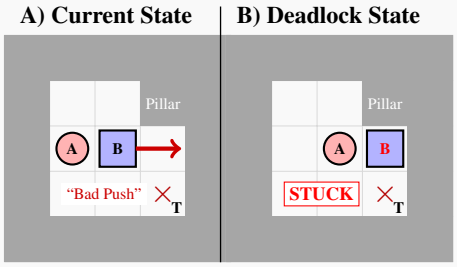
The effectiveness and stability of multi-turn RL depend critically on the properties of the *rollout distribution*, which directly controls the signal-to-noise ratio of advantage estimates, and which regions of the state-action space receive a learning signal (Schulman et al., 2015; 2017; Mroueh et al., 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. For example, consider a web agent that must purchase a specific item. Success requires a precise sequence of logical 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 is often 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. This motivates further search and heuristics to avoid such doomed prefixes (Ignatenko & Pravosud, 2023). To illustrate this, Fig. 1 shows how a single early mistake in Sokoban can render an episode unsalvageable, whereas simple 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.

Taken together, these issues suggest that beyond the choice of optimization algorithm (e.g., PPO/GRPO), the *quality of rollouts* is a central factor in multi-turn RL. Motivated by the example

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 (×), 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 unlike Best-of-N that avoids it.

in Fig. 1, we study whether allocating a modest, one-time increase in training-time compute during trajectory search can address these issues. Specifically, we formulate the following hypothesis:

Hypothesis: Per-Turn Rollout Optimization

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

We test this hypothesis by augmenting rollout generation via lightweight tree search guided by task feedback or proxy scores. This repurposes test-time search ideas for train-time data collection, analogous in spirit 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 making process that induces a search tree over possible action sequences (see Fig. 2). Starting from an initial state s_0, each node in the tree 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 branching factor. 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 general, $S(\cdot)$ is task-dependent and may simply correspond to the turn-level reward. However, it may also incorporate heuristic signals, learned value estimates, or proxy feedback if immediate rewards are unavailable. 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 proceeds by selecting actions according to a *search strategy* $\mathcal{F}_\phi$ parameterized by search parameters ϕ. Formally, we define:

$$\text{TSR} : (\pi_\theta, u, S, K, \mathcal{F}_\phi) \longrightarrow \{\tau_1, \ldots, \tau_M\} , \tag{3}$$

where $u \in \mathcal{U}$ denotes a task instance with initial state s_0, and $\mathcal{F}_\phi$ selects and composes actions by optimizing trajectory prefixes under the scoring function S over a finite horizon K. Here, the number of trajectories L generated by TSR is a hyper-parameter, which we omit from the input parameters for notational simplicity. By repeatedly expanding candidate actions and selecting according to $S(\cdot)$, rollout generation corresponds to traversing a truncated tree. The collected trajectories are then used in each policy optimization step. Crucially, TSR modifies only the *rollout generation process*, i.e., the policy 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 to improve a single output, here we adapt them to *training-time rollout generation* for multi-turn RL. Fig. 5 in App. C provides an overview of these three search methods.

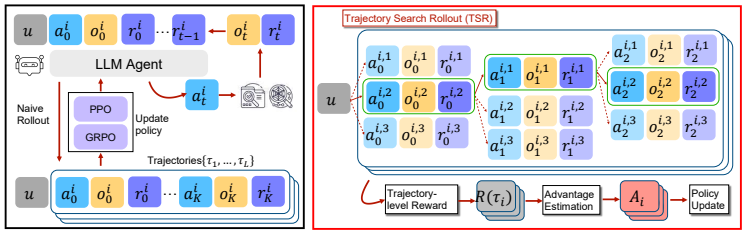


Figure 2: (**Left**) Multi-turn RL with naive rollouts: trajectories are sampled independently without any search. (**Right**) Trajectory Search Rollouts (TSR): lightweight tree-style search to construct high-quality trajectories by selecting high-scoring actions at each turn.

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 base policy π_θ, where each trajectory is typically scored using its cumulative reward $R(\tau)$. Rollout generation then retains only the best trajectory:

$$\tau^* = \arg \max_{\tau \in \{\tau^{(1)}, \ldots, \tau^{(N)}\}} R(\tau) \ . \tag{4}$$

However, 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 naive baseline of search with depth K and branching only at the root.

Per-Turn Beam Search. At each turn t, we sample a candidate action set $\mathcal{A}_t$ and score each candidate action a_t (along with its outcome o_t) using $S(\tau_{<t}, a_t, o_t)$. Instead of committing to a single action, beam search maintains a set of B active partial trajectories (beams), retaining the top-ranked prefixes after each expansion. Formally, if $\mathcal{B}_t$ denotes the set of active beams at turn t, then

$$\mathcal{B}_{t+1} = \mathrm{Top}_B \Big\{ (\tau_{<t} \circ a_t) \mid \tau_{<t} \in \mathcal{B}_t, \ a_t \in \mathcal{A}_t \Big\} \ , \tag{5}$$

where $\circ$ denotes trajectory concatenation and ranking is performed using accumulated or per-step scores. After K turns, the highest-scoring trajectory is selected for training. Per-turn beam search actively steers rollout generation toward promising regions of the trajectory space and enables recovery from locally suboptimal actions.

Shallow Lookahead Search. Similar to beam search, shallow lookahead evaluates candidate actions at each turn but additionally includes a short horizon of future steps (depth $D \ll K$) before selection. This corresponds to evaluating a truncated subtree rooted at $\tau_{<t}$ and choosing actions based on predicted downstream outcomes. Lookahead may improve quality relative to greedy scoring, while incurring less computational overhead than maintaining a full beam.

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, WebShop tasks, or problem statements). 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:

$$U(u; \pi_\theta) = \mathrm{Std}_{\tau \sim \pi_\theta(\cdot|u)} \big[R(\tau) \big] \ . \tag{6}$$

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: trajectories either solve trivial problems (uniformly high reward) or consistently fail (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 uncertainty-based 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 complementary methods is an interesting future work.

Algorithm 1 TSR with Instance-Level Filtering

Input: Policy π_θ, Task distribution $\mathcal{D}$, Sample Size P, Group Size N, Filter Ratio p, Search 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-$p\%$ subset $\mathcal{X}_{top}$.
  Update π_θ via PPO/GRPO using $\mathcal{X}_{top}$.
end for

4 Experimental Setup

To ensure reproducibility, we base our experiments on RAGEN (Wang et al., 2025), a popular framework for training and evaluating LLM agents in RL settings.

4.1 Environments and Tasks

We choose three diverse multi-turn environments from the RAGEN framework, selected to test agent performance under varying degrees of reasoning complexity:

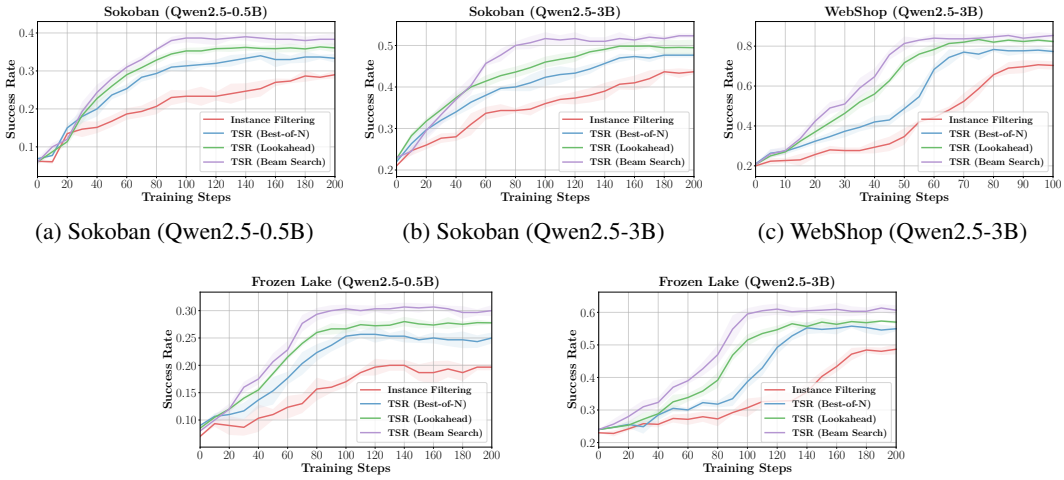
1. **Sokoban** (Weber et al., 2018). A logic puzzle in which the agent must push boxes into target locations. Feedback is deterministic and immediate, allowing to score per-turn rollouts via environment rewards.
2. **FrozenLake** (Brockman et al., 2016). A stochastic navigation task where the agent must cross a frozen lake to reach a goal while avoiding obstacles (holes in the ice). The iced surface is slippery such that the same action can stochastically lead to different next states. Rewards are extremely sparse, with a positive signal provided only upon reaching the goal.
3. **WebShop** (Yao et al., 2023a). 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 reward feedback in FrozenLake and WebShop, we employ task-specific proxy scores to guide rollout generation. Details and examples are provided in App. D.

4.2 Training and Baselines

For our main experiments, we train Qwen2.5-0.5B and Qwen2.5-3B instruct models (Qwen et al., 2025) 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 (Wang et al., 2025). 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) and $L = 16$ rollouts per group, with a maximum turn horizon of $K = 5$. Our baseline is instance-level filtering with naive rollout sampling using a filtering ratio of $p = 0.25$, similar to the default RAGEN implementation. When instantiating TSR with best-of-N, we sample $N = 28$ trajectories and retain the top $L = 16$. For beam search, we use a branching factor of $M = 4$ and beam width $B = 2$. For lookahead search, we use the same B and M with depth $D = 2$. We selected these parameters after performing ablations on N, M, B, and D (see App. F).

Note that after trajectory search rollout and filtering, all methods yield the same number of trajectories per group, ensuring a fair comparison. We provide additional details on our training setup 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 (↑), Average Response Length in tokens (↓), and Average Interaction Turns (↓) on the held-out validation set. **Bold** indicates the best performance within each environment and model size.

Task	Method	Qwen2.5-0.5B			Qwen2.5-3B		
		S.R. (↑)	Resp. Len (↓)	Turns (↓)	S.R. (↑)	Resp. Len (↓)	Turns (↓)
Sokoban	Base Model	8.9	293	4.7	16.0	272	4.6
	Instance Filtering	29.0	105	4.4	43.7	161	4.1
	TSR (Best-of-N)	33.3	103	4.3	47.7	165	4.0
	TSR (Lookahead)	36.1	101	4.0	49.5	158	3.8
	TSR (Beam Search)	**38.3**	**98**	**3.8**	**52.3**	**152**	**3.6**
FrozenLake	Base Model	6.4	276	4.6	12.7	225	3.9
	Instance Filtering	19.7	157	4.1	48.7	174	3.6
	TSR (Best-of-N)	25.0	182	4.2	55.0	161	3.5
	TSR (Lookahead)	27.8	168	3.8	57.0	135	3.3
	TSR (Beam Search)	**30.0**	**152**	**3.5**	**60.7**	**98**	**3.1**

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, consistent with the maximum turn horizon K. Besides success rate (an indicator for *task completion*), we report rollout entropy (an indicator for diversity (*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 (*token efficiency and task-completion latency*). Further details on evaluations are given in App. E.

5 Results and Discussion

We present key results and statistics assessing final task performance, training stability, and scaling behavior. We refer to App. F for supplementary ablations and results.

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.

Table 2: **Results (WebShop, Qwen2.5-3B).** Success Rate (↑), Average Response Length (↓), and Average Interaction Turns (↓) on the held-out validation set. **Bold** indicates the best results.

Method	Qwen2.5-3B		
	S.R. (↑)	Resp. Len (↓)	Turns (↓)
Base Model	3.0	747	7.7
Instance Filtering	70.3	519	6.8
TSR (Best-of-N)	77.3	504	6.5
TSR (Lookahead)	82.3	475	6.1
TSR (Beam Search)	**85.3**	**453**	**5.8**

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 8.4% and 12.0% over instance filtering. Similarly, TSR (Lookahead) and TSR (Best-of-N) achieve improvements ranging from 4–12%. The largest gains are on WebShop, where best-of-N, lookahead, and beam search achieve performance improvements of 7.0%, 12.3%, and 15.0%, respectively, demonstrating that TSR yields strong benefits in long-horizon settings.

Among TSR variants, beam search consistently achieves the strongest performance and converges faster (see Fig. 3), followed by lookahead and best-of-N. Overall, we have three takeaways. First, beam search maintains multiple partial trajectories, enabling recovery from early irreversible mistakes and providing robustness to proxy-scoring noise and environment stochasticity, particularly in FrozenLake and WebShop. Second, lookahead commits to a single action based on proxy signals of future outcomes. While effective in deterministic settings like Sokoban, its gains are constrained by proxy quality and diminish as search depth increases. Third, 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.

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.

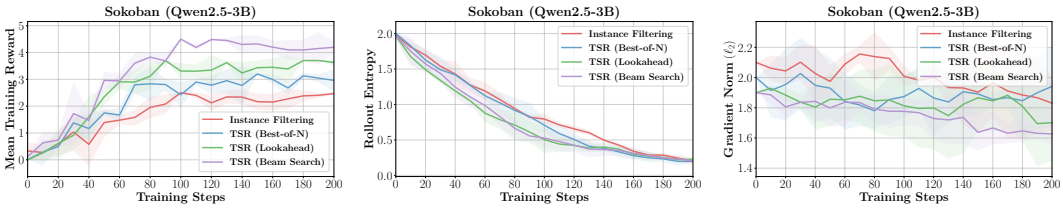
For exploitation, we analyze average training rewards in Fig. 4a. Across all TSR variants, average rewards increase steadily over training, indicating effective learning and policy improvement. We see trends consistent with overall task performance: beam search has on average higher rewards, followed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

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 trajectory search rollout does not alter or destabilize exploration–exploitation dynamics.

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 increased training-time compute can improve solution quality, token efficiency, and latency at inference time. Across environments, 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 train-time trajectory search not only improves task success but also yields more concise interactions at deployment time. We substantiate these results with qualitative examples for WebShop in App. F.1.



(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.

Table 3: **Comparison to Larger Models.** Qwen2.5-0.5B trained via TSR outperforms GPT-4o and Qwen2.5-72B models.

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

Method	Search Budget	Succes Rate	Δ
TSR (Beam Search)	$M = 2, B = 1$	50.6	–
	$M = 2, B = 2$	52.3	+1.7
	$M = 6, B = 2$	53.4	+1.1

5.4 Comparison to Larger Models

We benchmark our TSR-trained Qwen2.5-0.5B agents against GPT-4o (Hurst et al., 2024) and Qwen2.5-72B (Qwen et al., 2025) for Sokoban and FrozenLake in Table 3. Despite being orders of magnitude smaller, TSR (Beam Search) outperforms GPT-4o by 10.6% on Sokoban and 3.4% on FrozenLake. This suggests that multi-turn RL with TSR can yield performance that surpasses significantly larger, state-of-the-art models.

5.5 Scaling Search Budget vs. Performance

We further study how increasing training-time search budgets affects final performance. As shown in Table 4, we observe diminishing marginal returns. For beam search, the largest improvement ($+1.7\%$) comes from increasing the beam width from $B = 1$ to $B = 2$, enabling the agent to escape local optima. Further increasing the number of action samples from $M = 2$ to $M = 6$ yields only modest gains ($+1.1\%$). We observe similar behavior for other TSR variants. We provide more detailed scaling experiments for beam search in App. F.2.

6 Conclusion

In this work, we introduced Trajectory-Search Rollouts (TSR), a framework that repurposes inference-time search algorithms, such as beam search, lookahead, and best-of-N, to enhance the training data quality of policy gradient methods in multi-turn agentic RL. By actively exploring and pruning trajectories on a per-turn-level during the rollout phase, TSR overcomes the exploration challenges inherent in sparse- or delayed-reward environments and substantially increases final task accuracy, while effectively preventing mode collapse. Our methodology is simple, general, and optimizer-agnostic, and can thus be paired with any policy gradient method in principle. Empirically, we demonstrate that a 0.5B model trained with TSR not only significantly outperforms strong RL baselines but also surpasses larger generalist models on three representative tasks and environments. Our findings thus confirm that allocating additional compute at the trajectory rollout stage improves performance and stability in multi-turn RL training.

ACKNOWLEDGEMENTS

This work was supported in part by the German Federal Ministry of Research, Technology and Space (BMFTR) within the research hub 6G-life (Grant 16KISK002) and through the project AISAC (Grant Number 16KIS2462), by the Bavarian Ministry of Science and the Arts and the Saxon Ministry for Science, Culture, and Tourism through the project Next Generation AI Computing (gAIn), by the Bavarian Ministry of Economic Affairs, Regional Development and Energy through the project 6G Future Lab Bavaria, and in part by IBM Research.

REFERENCES

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.

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.

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.

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.

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.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. GPT-4o System Card. *arXiv preprint arXiv:2410.21276*, 2024.

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.

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

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.

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.

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.

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.

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.

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.

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

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.

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

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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, 2025a. URL https://arxiv.org/abs/2506.12928.

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

A Related Works

We present a short overview of relevant related works.

Multi-Turn Agent RL Frameworks. There are many practical frameworks to train multi-turn RL agents in diverse environments. For example, Agent Lightning (Luo et al., 2025) and AgentGym-RL (Xi et al., 2025a) provides decoupled training systems and modular pipelines, building on AgentGym (Xi et al., 2024) which only primarily focuses on environment diversity and data collection. SkyRL (Cao et al., 2025) offers a full-stack library for tool-use agents, utilizing asynchronous weight updates to maximize troughput. Similarly, slime (Zhu et al., 2025b) decouples training from rollouts by connecting Megatron with SGLang, specifically targeting high-performance scaling for multi-turn RL. RAGEN/StarPO (Wang et al., 2025) provides a trajectory-level RL formulation and documents instability patterns; StarPO-S (Wang et al., 2025) mitigates collapse via instance filtering (retain top-p% most-uncertain prompts, measured by reward-std across repeated rollouts) and gradient shaping. Agent Lightning (Luo et al., 2025) and AgentGym-RL (Xi et al., 2025a) provide decoupled training systems and modular pipelines across environments and RL algorithms. TSR is orthogonal: we upgrade the rollout generator with per-turn search; the optimizer, loss, and framework remain unchanged. We also adopt instance filtering from (Wang et al., 2025) as a separate stabilizer in our experiments, but our key novelty is trajectory search within the training rollout loop.

Test-time scaling Prior work has investigated various inference-time scaling strategies such as Best-of-N (BoN) sampling (Ouyang et al., 2022; Gui et al., 2024), beam search (Wang et al., 2024; Alzorgan & Razi, 2025), and shallow lookahead (Lefebvre & Durand, 2025). These approaches demonstrate compute-optimal allocation at inference, showing reliable gains under fixed extra compute. TSR transfers this idea to training-time rollouts: search once (during training), benefit many (during inference), and optionally combine with light test-time search if desired. Our instantiations follow BoN, beam search, and shallow lookahead without claiming compute-optimal allocation.

Rejection-sampling-style selection Several single-turn training works perform response-level filtering. For example, GFPO (Shrivastava et al., 2025) filters by response length or reward-per-token within GRPO-style updates. On the other hand, RAFT-style methods (Dong et al., 2023) train on positively rewarded samples. TSR differs on two axes: (i) multi-turn agent RL with interactive environments, and (ii) trajectory-level, per-turn tree search to construct candidate paths before selection. We view these lines as complementary and compare to search-free selection baselines under matched compute.

Test-time scaling for multi-turn RL Recent studies have extended inference scaling to agents by identifying "compute-optimal" frontiers for agents by balancing search breadth against environmental feedback depth (Zhu et al., 2025a). This includes scaling external interactions and backtracking to compensate for base policy reasoning limits (Shen et al., 2025). Unlike these inference-only approaches, TSR shifts search-intensive compute to the training phase, distilling high-quality trajectories into the model weights to reduce test-time latency.

Turn-level credit assignment Recent work addresses sparse trajectory rewards by extending PPO and GRPO to multi-turn variants (MT-PPO/MT-GRPO) by introducing turn-level reward designs (Wei et al., 2025), enabling immediate feedback via verifiable signals or LLM-judging. Similarly, the AgentPRM framework (Xi et al., 2025b) utilizes process reward models to provide step-wise evaluations for long-horizon tasks. While these methods refine the learning signal, TSR focuses on trajectory generation, making it complementary for multi-turn stability.

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.

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:

$$J_{\text{PPO}}(\theta) = \mathbb{E}_{\tau \sim \pi_{\theta_{\text{old}}}} \left[\frac{1}{K} \sum_{t=0}^{K-1} \min \left(r_t(\theta) \hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon) \hat{A}_t \right) \right], \tag{7}$$

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:

$$\hat{A}_i = \frac{R_i - \text{mean}(R_1, \ldots, R_G)}{\text{std}(R_1, \ldots, R_G) + \epsilon}. \tag{8}$$

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

$$J_{\text{GRPO}}(\theta) = \mathbb{E}_{\{\tau_i\}_{i=1}^{G} \sim \pi_{\theta_{\text{old}}}} \left[\frac{1}{G} \sum_{i=1}^{G} \frac{1}{K_i} \sum_{t=0}^{K_i-1} \min \left(r_{i,t}(\theta)\, \hat{A}_i,\ \text{clip}(r_{i,t}(\theta), 1-\epsilon, 1+\epsilon)\, \hat{A}_i \right) \right], \tag{9}$$

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.

C Trajectory Search Rollout Search Strategies

We instantiate our TSR framework with concrete search strategies inspired by test-time scaling (Snell et al., 2024), including best-of-N, beam search, and shallow lookahead.

Best-of-N. From a given Sokoban state, the policy samples N complete rollouts independently. Trajectories that commit early to irreversible moves (e.g., pushing a box into a dead-end) terminate with low return, while trajectories that preserve maneuverability succeed. TSR retains only the highest-scoring trajectory for training.

Beam Search. At each turn, beam search maintains multiple partial Sokoban trajectories and expands them in parallel. Prefixes that lead toward dead-end configurations are pruned once their future feasibility collapses, allowing the search to continue only along branches that remain solvable.

Shallow Lookahead. For each candidate action in the current Sokoban state, shallow lookahead briefly simulates a few future steps before committing. Actions whose short-horizon continuations quickly lead to blocked or low-mobility states are avoided in favor of actions that preserve access for subsequent box pushes.

Fig. 5 provides a conceptual overview of best-of-N, beam search, and shallow lookahead search strategies applied in the multi-turn RL setting.

D RL Environments

This section provides a detailed description of the Sokoban, FrozenLake, and WebShop environments used throughout our experiments, as well as the derivation of task-specific proxy scores to assess per-turn interactions during multi-turn rollout generation.

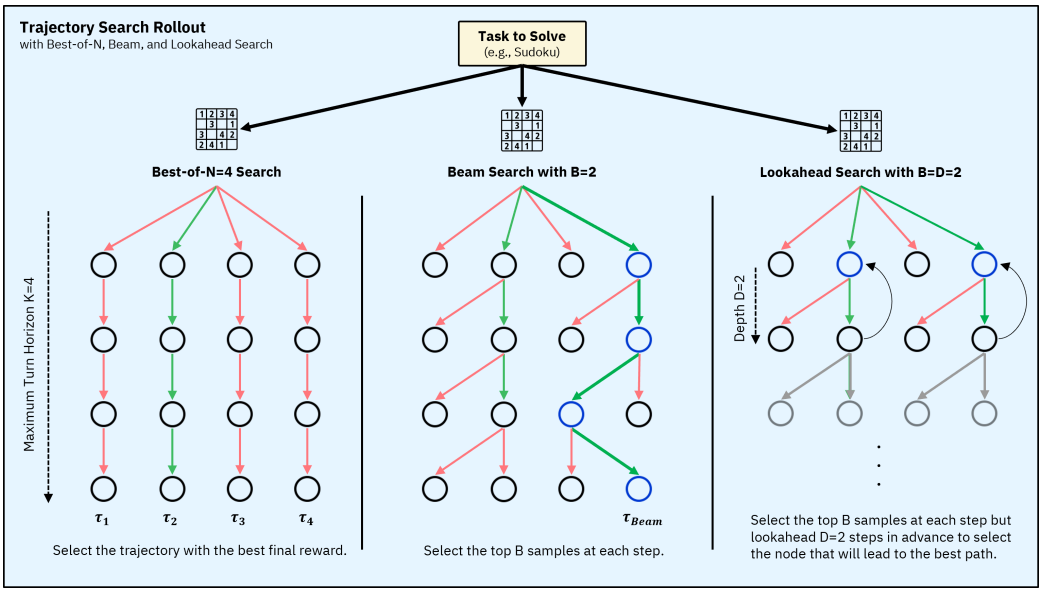


Figure 5: Illustration of TSR tree-search–based rollout generation with best-of-N, beam search, and shallow lookahead search methods.

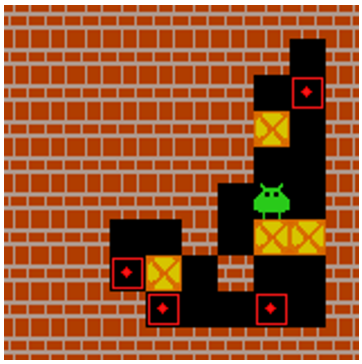


Figure 6: A sample Sokoban environment.

Figure 7: A sample FrozenLake environment.

D.1 Environment Descriptions

Sokoban (Weber et al., 2018) A determinist logic puzzle where the agent is tasked with pushing a set of boxes into designated target locations within a grid-based warehouse. The task requires the agent to navigate the grid and interact with boxes via orthogonal pushes, constrained by the inability to pull the boxes or push multiple boxes simultaneously. The objective is to reach a state where all boxes are positioned on the target locations, requiring the agent to navigate the grid and avoid deadlocks where the blocks are pushed into a corner. Rewards are immediate and rolled out per turn with focus on 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 Figure 6.

FrozenLake (Brockman et al., 2016) A stochastic navigation game where the agent is tasked with crossing a grid-based frozen lake from a starting point to a goal location. The environment consists of a grid containing safe frozen tiles, lethal holes, and a terminal goal tile. The task requires the agent to navigate using orthogonal movements, constrained by the potential for stochastic transitions where the agent may slide in a direction unintended due to the slippery ice. Each movement action succeeds with probability 1/3 and deviates perpendicularly with probability 2/3. Rewards are sparse and rolled out upon reaching the goal: +1 on successful completion and 0 for all others. A sample environment is shown in Figure 7.

WebShop (Yao et al., 2023a) A determinist e-commerce scenario where the agent is tasked with finding and purchasing specific products on a simulated e-commerce platform. The environment consists of over a million real-world products, categorized by attributes, descriptions, and pricing. The task requires the agent to navigate the website by executing high-level actions such as searching, clicking on product links, and selecting options. The agent is constrained by the need to match a natural language instruction that specifies multiple product constraints. The objective is to reach the final checkout state with the correct item, requiring the agent to effectively filter irrelevant search results and accurately map user requirements to available product specifications.

D.2 Task-Specific Score Design

Unlike Sokoban, per-turn rollout generation in FrozenLake and WebShop cannot be effectively guided by environment rewards, as rewards are either extremely sparse or delayed. In these settings, intermediate reward signals are uninformative or entirely unavailable, rendering direct reward-based scoring ineffective. Instead, we employ task-specific proxy scores that encode progress and safety, enabling TSR to rank candidate actions during rollout generation. All proxy scores are lightweight and do not require training an auxiliary value function.

FrozenLake. FrozenLake exhibits stochastic transition dynamics and provides a non-zero reward only upon reaching the goal. To guide rollout generation, we define a heuristic 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. We define the score function

$$S_{\text{FL}}(s_t, a_t) = -d(s_{t+1}, g) - \lambda \cdot \mathbb{I}[s_{t+1} \in \mathcal{H}], \tag{10}$$

where $d(\cdot, \cdot)$ denotes the Manhattan distance on the grid, $\mathbb{I}[\cdot]$ is the indicator function, and $\lambda > 0$ controls the penalty for transitioning into a hole. This score favors actions that reduce distance to the goal while strongly penalizing unsafe transitions. Despite stochastic dynamics, this heuristic provides a meaningful local signal for ranking candidate actions during rollout construction.

WebShop. WebShop is deterministic but characterized by delayed rewards, as success is only signaled after a correct purchase at episode termination. Consequently, per-turn environment rewards are unavailable during rollout generation. Similar to the original WebShop implementation (Yao et al., 2023a), we define a proxy score based on task progress and instruction–page alignment. Let u denote the user instruction and p_t the textual content of the current webpage. We define

$$S_{\text{WS}}(h_t, a_t) = \alpha \cdot \mathbb{I}[\text{progress}(h_{t+1})] + \beta \cdot \text{sim}(u, p_{t+1}), \tag{11}$$

where progress$(\cdot)$ is a binary indicator capturing key milestones (e.g., reaching a product page, adding an item to the cart, entering checkout), and sim$(\cdot, \cdot)$ denotes a lightweight textual similarity measure between the instruction and the current page (e.g., keyword overlap). The weights $\alpha, \beta > 0$ balance structural progress and semantic alignment. This proxy score provides a weak but informative signal for guiding long-horizon rollout generation.

E Training and Evaluation Setup

Our training setup closely follows the experimental configuration of RAGEN (Wang et al., 2025), and we therefore only 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 RAGEN framework.

Models and Optimization. We initialize all agents from Qwen2.5-Instruct 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 RAGEN'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), each generating $N = 16$ rollouts, resulting in 256 trajectories per iteration. Each trajectory is limited to a maximum of 5 interaction turns, with up to 5 actions per turn and at most 10 total actions per episode.

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) 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 $T = 0.5$, and episodes are truncated after 5 turns or 10 total actions. We report the following metrics:

1. **Success Rate / Final Task Accuracy.** Task completion rate 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 a proxy for exploration and policy uncertainty. Sharp drops 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), to measure whether TSR reduces interaction budget after training.
6. **Wall-Clock Time.** Total training time per run (on 8 NVIDIA A100 GPUs), reported to quantify the computational overhead introduced by search-based rollout generation.

F Supplementary Results

In this section, we provide additional ablations and results.

F.1 Inference Efficiency

We provide a qualitative example of inference efficiency for the WebShop task, where we show a typical failure mode of the baseline agent (looping / redundant actions) and a more direct rollout from TSR-beam.

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 latte, 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.*)

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]
Reward: 0.0

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

F.2 Scaling Search Budgets

We provide additional experiments analyzing the effect of scaling search budget parameters for TSR (Beam Search) in Tables 5, 6, and 7 for Sokoban, FrozenLake, and WebShop, respectively. The results corroborate our main findings and show that the largest gains arise from increasing the beam width from $B = 1$ to $B = 2$. In contrast, increasing the number of action samples M exhibits diminishing returns, leading to a plateauing performance trend. Overall, we find that moderate search budgets (e.g., $B = 2$, $M = 4$) offer a favorable trade-off between performance and computational cost. Similar trends are observed for TSR (Best-of-N) and TSR (Lookahead) variants, which we omit for brevity.

Table 5: **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		
	S.R. (↑)	Resp. Len (↓)	Turns (↓)	S.R. (↑)	Resp. Len (↓)	Turns (↓)
M=2, B=1	36.8	101	3.95	50.6	156	3.70
M=4, B=1	37.6	99	3.88	51.4	154	3.65
M=6, B=1	37.9	98	3.85	51.8	153	3.62
M=2, B=2	38.3	98	3.80	52.3	152	3.60
M=4, B=2	39.1	96	3.75	53.1	150	3.55
M=6, B=2	**39.4**	**95**	**3.72**	**53.4**	**149**	**3.52**

F.3 Wall-Clock Time Analysis

Shifting trajectory search from inference time to training time incurs additional computational overhead during training. While part of this cost can be amortized at inference time through improved

Table 6: **FrozenLake: Beam Search Scaling.** Performance trends are consistent with Sokoban, where wider beams ($B = 2$) outperform greedy search ($B = 1$) across all sample counts.

Budget	Qwen2.5-0.5B			Qwen2.5-3B		
	S.R. (↑)	Resp. Len (↓)	Turns (↓)	S.R. (↑)	Resp. Len (↓)	Turns (↓)
M=2, B=1	28.7	160	3.65	58.9	110	3.20
M=4, B=1	29.3	157	3.58	59.6	106	3.16
M=6, B=1	29.5	156	3.56	59.9	104	3.14
M=2, B=2	30.0	152	3.50	60.7	98	3.10
M=4, B=2	30.6	150	3.47	61.3	96	3.08
M=6, B=2	**30.8**	**149**	**3.46**	**61.5**	**95**	**3.07**

Table 7: **WebShop: Beam Search Scaling (Qwen2.5-3B).** Performance trends are consistent with Sokoban and FrozenLake, where wider beams ($B = 2$) outperform greedy search ($B = 1$).

Budget	Qwen2.5-3B		
	S.R. (↑)	Resp. Len (↓)	Turns (↓)
M=2, B=1	83.6	470	6.05
M=4, B=1	84.5	462	5.90
M=6, B=1	84.8	458	5.86
M=2, B=2	85.3	453	5.80
M=4, B=2	86.0	448	5.75
M=6, B=2	**86.3**	**446**	**5.72**

efficiency (e.g., fewer interaction turns and shorter responses), it remains important to analyze the overall wall-clock cost and convergence behavior.

Tables 8, 9, and 10 report wall-clock training times for the Sokoban, FrozenLake, and WebShop tasks, respectively, under the search configurations used in the main body. In particular, we report wall-clock time (in hours), the number of training steps and elapsed time required to reach 80% of the final success rate, and the corresponding steps and wall-clock time to final convergence.

In general, trajectory search rollout incurs more training-time overhead compared to the instance-level filtering baseline. However, when considering both time-to-convergence metrics and the learning dynamics shown in Fig. 3, we observe that trajectory search rollout methods typically converge in fewer training iterations. As a result, the total time to reach convergence remains comparable to the instance-level filtering baseline across tasks and model sizes, despite the higher per-iteration cost.

Overall, these results indicate that TSR does not significantly slow down end-to-end training. Instead, they trade increased per-step computation for faster convergence and improved final performance, yielding similar wall-clock time to convergence while achieving higher accuracy.

Table 8: Wall-clock time and convergence analysis on **Sokoban** (Qwen2.5-0.5B and Qwen2.5-3B). We report final success rate, total wall-clock training time, and the number of steps and wall-clock time required to reach 80% of final performance and full convergence.

Model	Method	Final Success Rate (%)	Wall Time (h)	Steps to 80%	Time to 80% (h)	Steps to Convergence	Time to Convergence (h)
Qwen2.5-0.5B	Instance Filtering	29.0	**1.72**	90	0.77	180	1.54
	TSR (Best-of-N)	33.3	2.51	60	**0.75**	120	**1.51**
	TSR (Lookahead)	36.1	3.11	55	0.86	105	1.63
	TSR (Beam Search)	**38.3**	3.69	**50**	0.92	**90**	1.66
Qwen2.5-3B	Instance Filtering	43.7	**2.22**	100	1.11	180	**1.99**
	TSR (Best-of-N)	47.7	3.37	70	1.18	150	2.53
	TSR (Lookahead)	49.5	3.96	45	**0.89**	150	2.96
	TSR (Beam Search)	**52.3**	4.56	**55**	1.25	**90**	2.05

Table 9: Wall-clock time and convergence analysis on **FrozenLake** (Qwen2.5-0.5B and Qwen2.5-3B). We report final success rate, total wall-clock training time, and the number of steps and wall-clock time required to reach 80% of final performance and full convergence.

Model	Method	Final Success Rate (%)	Wall Time (h)	Steps to 80%	Time to 80% (h)	Steps to Convergence	Time to Convergence (h)
Qwen2.5-0.5B	Instance Filtering	19.7	**1.34**	80	**0.54**	120	**0.80**
	TSR (Best-of-N)	25.0	2.64	70	0.93	100	1.32
	TSR (Lookahead)	27.8	2.53	**60**	0.76	90	1.14
	TSR (Beam Search)	**30.0**	2.38	**60**	0.71	**80**	0.95
Qwen2.5-3B	Instance Filtering	48.7	**1.85**	150	1.39	170	1.57
	TSR (Best-of-N)	55.0	3.47	110	1.91	140	2.43
	TSR (Lookahead)	57.0	3.19	90	1.44	130	2.08
	TSR (Beam Search)	**60.7**	2.90	**80**	**1.16**	**100**	**1.45**

Table 10: Wall-clock time and convergence analysis on **WebShop** (Qwen2.5-3B). We report final success rate, total wall-clock training time, and the number of steps and wall-clock time required to reach 80% of final performance and full convergence.

Method	Final Success Rate (%)	Wall Time (h)	Steps to 80%	Time to 80% (h)	Steps to Convergence	Time to Convergence (h)
Instance Filtering	70.3	**6.24**	78	**4.87**	85	**5.31**
TSR (Best-of-N)	77.3	9.75	60	5.85	70	6.83
TSR (Lookahead)	82.3	12.78	**45**	5.75	65	8.31
TSR (Beam Search)	**85.3**	14.94	**45**	6.73	**50**	7.47