# Process Reward Informed Tree Rollout for Effective Multi-Turn RL

**Xintong Li**[1*], **Sha Li**[2], **Yuwei Zhang**[1], **Changlong Yu**[2], **Rongmei Lin**[2], **Hongye Jin**[2], **Shuyi Guan**[3], **Xin Liu**[2], **Linwei Li**[2], **Qingyu Yin**[2], **Jingbo Shang**[1]
[1]UC San Diego [2]Amazon [3]MIT Alumni
{xil240,jshang}@ucsd.edu slliz@amazon.com

arXiv:2607.15610v1 [cs.CL] 17 Jul 2026

## Abstract

Reinforcement learning (RL) has become a key approach for training LLM agents, yet popular methods such as GRPO/RLOO rely on multiple independently sampled complete trajectories for advantage estimation. In long-horizon agentic tasks, such a uniform rollout strategy can waste budget on uninformative dead-end attempts, while promising intermediate states do not receive sufficient exploration. The multi-turn structure of agentic trajectories, with interleaved actions and observations, naturally supports organizing a trajectory group as a tree, where each turn serves as a decision point for exploration. This perspective reframes effective exploration as the problem of deciding where to branch. We propose Process-Scorer Guided Adaptive Tree Rollout (PATR), a quality-aware rollout framework for multi-turn agent RL. PATR uses task-appropriate process feedback to score partial trajectories, selectively branches from promising states, reuses shared prefixes, and conservatively stops degenerate paths to reduce wasted sampling. The resulting rollout groups remain compatible with standard policy optimization while providing more efficient exploration under the same training budget. We evaluate PATR on FrozenLake and the challenging SWE-Bench, which is largely unexplored by prior tree-rollout agent RL methods. Experiments show that PATR improves performance by up to $+5.0$ points on SWE-Bench and $+9.3$ points on FrozenLake, highlighting process-guided tree rollouts as an effective strategy for scalable multi-turn RL.

## 1 Introduction

Large language models (LLMs) are increasingly trained as agents that solve tasks through multi-turn interactions with tools, feedback, and external observations (Yao et al., 2022; Schick et al., 2023; Shinn et al., 2023). Reinforcement learning (RL) such as GRPO (Shao et al., 2024) has become a standard approach for improving such agentic capabilities. However, most existing GRPO-style agent training still relies on independently sampled complete trajectories (Li et al., 2026a). This rollout strategy allocates computation uniformly across trajectories, regardless of their intermediate quality. For long-horizon agentic tasks, such uniform allocation can waste budget on uninformative trajectories, such as repeated tool-use loops, while promising intermediate states receive insufficient exploration. As a result, the rollout group may contain redundant or low-value trajectories, leading to inefficient exploration and noisy advantage estimation.

Recent work has begun to improve rollout generation through tree-structured sampling, dynamic branching, and step-level feedback. In single-turn reasoning, tree-based methods reuse shared prefixes and explore multiple continuations from intermediate states, showing that rollout construction itself can be an important component of RL training (Surana et al., 2026; Xing et al., 2025). This idea is even more critical for multi-turn agents, where each action changes the future context and final rewards are often sparse or delayed (Feng et al., 2026; Djuhera et al., 2026).

Nevertheless, existing multi-turn tree rollout methods exhibit key limitations in how they guide exploration. One line of work branch around high-entropy tool-use steps (Dong et al., 2025b,a) or allocate rollout budget using value-based uncertainty estimates (Cao et al., 2026). While these signals encourage exploration, they do not directly measure whether a partial trajectory is making meaningful progress toward the task objective. Another line of work applies per-turn search to select the highest-scoring action at every step (Djuhera et al., 2026), which can bias the rollout distribution away from the current policy and discard informative negative examples. These limitations motivate a rollout mechanism that allocates budget based on process-

[*]Work done during internship at Amazon.

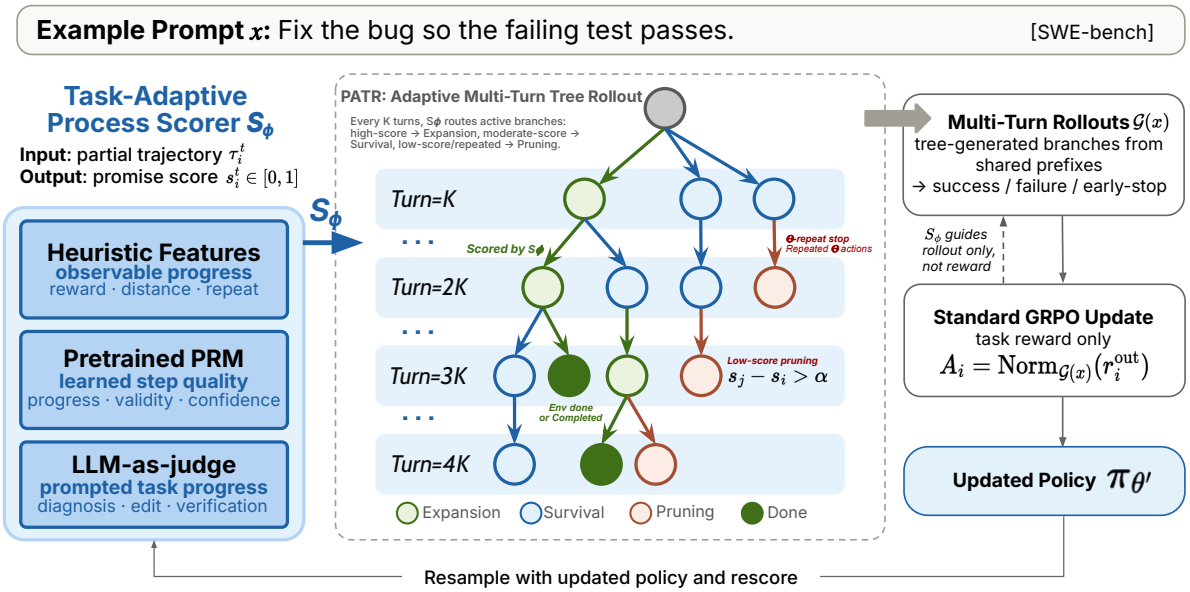


Figure 1: Overview of PATR. Task-adaptive process scorer evaluates partial trajectories and guides adaptive rollout through expansion, survival, and pruning. The rollout group is optimized with standard GRPO using task rewards.

level trajectory quality while preserving diverse outcomes, including failures, for policy learning.

We propose Process-Scorer Guided Adaptive Tree Rollout (PATR), a cost-effective rollout generation framework for multi-turn agent RL that allocates more rollout budget to promising partial trajectories while terminating unpromising ones early. For each training instance, PATR constructs a tree of partial trajectories instead of sampling independent full trajectories from scratch. We periodically evaluate active branches by a task-adaptive process scorer and route into one of three paths: high-scoring branches are *expanded* by sampling multiple child continuations from the same intermediate state, moderate-scoring branches *survive* into the next iteration, and low-scoring or degenerate branches are *pruned* early. Pruned branches are retained rather than discarded, preserving rollout diversity and serving as negative examples for policy optimization. Since child branches inherit their parent prefixes, PATR naturally reuses shared computation and produces more informative rollout trajectories under a comparable sampling budget. We then employ standard GRPO-style policy optimization on the resulting rollout group with the same outcome rewards, avoiding direct optimization against the auxiliary process scorer.

We evaluate PATR on both FrozenLake and the more challenging SWE-Bench. FrozenLake provides a controlled testbed where process quality can be estimated with simple progress heuristics, while SWE-Bench evaluates whether process-guided tree rollouts can support realistic long-horizon coding-agent tasks. Experiments show that PATR improves performance by up to $+5.0$ points on SWE-Bench and $+9.3$ points on FrozenLake, highlighting the value of process-guided rollout construction for effective multi-turn RL.

Our contributions are summarized as follows:

- We propose PATR, a process-guided adaptive tree rollout framework that expands promising partial trajectories, preserves surviving branches, and prunes degenerate paths.
- We keep PATR compatible with GRPO by using process feedback only for rollout allocation while optimizing the policy with standard outcome rewards.
- We validate PATR on FrozenLake and SWE-Bench, showing consistent gains over GRPO and other rollout-generation baselines.

## 2 Related Work

**Adaptive Rollout Generation for Agent RL.** GRPO (Shao et al., 2024) and subsequent RL-based reasoning and agent methods, such as DeepSeek-R1 (Guo et al., 2025) and ARPO (Dong et al., 2025b), commonly sample multiple trajectories per task and compute group-relative advantages from outcome rewards. Recent work improves this rollout process through structured or adaptive sampling. ARPO (Dong et al., 2025b) and

AEPO (Dong et al., 2025a) branch around high-entropy tool-use steps, ATPO (Cao et al., 2026) allocates rollout budget using uncertainty estimates in medical dialogue, and TSR (Djuhera et al., 2026) applies tree-style search during training-time rollout generation. Tree-based reasoning methods such as Tree of Thoughts (Yao et al., 2023), RAP (Hao et al., 2023), and tree-search agents (Koh et al., 2024) also explore multiple continuations, but mainly for inference-time solution selection. Our work follows the training-time rollout perspective, but uses process-level quality scores rather than entropy or uncertainty alone to guide branch expansion.

**Process Feedback and Step-Level Supervision.** Process reward models, introduced in step-by-step verification work such as PRM (Lightman et al., 2024), Math-Shepherd (Wang et al., 2024), and Skywork PRM (He et al., 2025), provide intermediate quality estimates for multi-step reasoning. Several methods incorporate such signals into training objectives: PRIME (Cui et al., 2025) derives implicit process rewards, iStar (Liu et al., 2025) learns step-level rewards during agent training, WS-GRPO (Mundada et al., 2026) constructs weakly supervised prefix rewards, and Step-GRPO (Li et al., 2026b) modifies GRPO with structured process supervision. LLM-as-judge methods (Zheng et al., 2023) offer a flexible alternative when task-specific PRMs are unavailable, especially for open-ended agentic tasks where intermediate progress is difficult to specify manually. In contrast, our method uses process feedback only to allocate rollout budget, leaving the policy objective and outcome reward unchanged.

## 3 Preliminaries

### 3.1 Multi-Turn Agentic Reinforcement Learning

We consider reinforcement learning for language agents that solve tasks through multi-turn interaction with external systems. Given a task prompt $x$, after $t$ turns the agent observes a history $h_t = (x, a_1, o_1, \ldots, a_t, o_t)$, where $a_t$ is the agent action and $o_t$ is the corresponding observation. The policy samples the next action as $a_{t+1} \sim \pi_\theta(\cdot \mid h_t)$. A completed rollout $\tau = (x, a_1, o_1, \ldots, a_T, o_T)$ receives an outcome reward $r^{\text{out}}(\tau)$ upon termination. In agentic tasks, this reward is often sparse or delayed, making rollout quality critical for policy optimization (Wang et al., 2025).

### 3.2 Group-Relative Policy Optimization

GRPO (Shao et al., 2024) samples a group of $G$ rollouts for each task and computes group-normalized rewards, avoiding the need for a learned value function. For trajectory $\tau_i$ in the rollout group of task $x$, the outcome advantage is,

$$A_i^{\text{out}} = \frac{r_i^{\text{out}} - \mu_{\text{out}}(x)}{\sigma_{\text{out}}(x) + \varepsilon}, \tag{1}$$

where $r_i^{\text{out}} = r^{\text{out}}(\tau_i)$, and $\mu_{\text{out}}(x)$ and $\sigma_{\text{out}}(x)$ are the mean and standard deviation of outcome rewards within the group.

The policy is optimized with the clipped importance-weighted objective

$$\mathcal{L}(\theta) = \mathbb{E}_x \left[ \frac{1}{G} \sum_{i=1}^{G} \frac{1}{|\tau_i|} \sum_{t=1}^{|\tau_i|} \min\Big( \rho_{i,t}(\theta) A_i^{\text{out}}, \; \text{clip}\big(\rho_{i,t}(\theta), 1-\epsilon, 1+\epsilon\big) A_i^{\text{out}} \Big) \right], \tag{2}$$

where $\rho_{i,t}(\theta) = \frac{\pi_\theta(a_{i,t} | h_{i,t-1})}{\pi_{\theta_{\text{old}}}(a_{i,t} | h_{i,t-1})}$ is the importance ratio. Since advantages are computed relative to the sampled group, policy learning depends directly on rollout-group quality. However, standard GRPO samples complete trajectories independently from the initial task state, without considering intermediate trajectory quality.

## 4 PATR: Process-Scorer Guided Adaptive Tree Rollout

We propose **PATR**(Process-scorer guided Adaptive Tree Rollout), a cost-effective rollout generation framework for multi-turn agent RL, that allocates more rollout budget to promising partial trajectories and terminate unpromising ones early, producing groups with diverse outcomes from shared prefixes. Given a task prompt $x$, PATR constructs a tree of partial trajectories and adaptively allocates rollout budget using an intermediate process scorer. The full procedure is summarized in Algorithm 1.

### 4.1 Adaptive Tree Construction

PATR initializes each task prompt $x$ with $B_0$ active branches sampled from the current policy. Let $\mathcal{A}$ and $\mathcal{C}$ denote the sets of active and completed branches, respectively. At each iteration, every active branch is advanced for up to $K$ interaction steps. Branches that terminate during this interval, such as by successful completion or reaching the maximum step budget, are moved from $\mathcal{A}$ to

$\mathcal{C}$. For each remaining active branch $\tau_i^t$ at step $t$ ($t \in \{K, 2K, 3K, \ldots\}$), a process scorer $S_\phi$ produces a scalar score

$$s_i^t = S_\phi(x, \tau_i^t) \in [0, 1], \tag{3}$$

where higher scores indicate more promising partial trajectories for further exploration.

After scoring, each active branch is routed into one of three paths based on its process score: **expansion**, **survival**, or **pruning**.

**Expansion.** High-scoring branches are selected into an expansion set $\mathcal{E} \subseteq \mathcal{A}$. For each selected branch $\tau_i^t$, PATR samples $M$ child continuations from the same intermediate state:

$$\tau_i^{t,m} \sim \pi_\theta(\cdot \mid x, \tau_i^t), \qquad m = 1, \ldots, M. \tag{4}$$

These children share the parent prefix and explore different continuations, allowing PATR to allocate additional samples to promising intermediate states. The parent branch is then removed from $\mathcal{A}$ and replaced by its children.

**Survival.** Branches with moderate scores remain active. They are advanced for another $K$ steps in the next iteration and re-evaluated with updated process scores.

**Pruning.** PATR terminates a branch $\tau_i^t$ when its process score is substantially lower than the current active set:

$$s_i^t < \text{median}_{j \in \mathcal{A}}\, s_j^t - \alpha, \tag{5}$$

where $\alpha$ is a margin threshold. PATR also stops deterministically degenerate branches, such as those repeating the same action for $\Theta$ consecutive turns. Early-stopped branches are moved to $\mathcal{C}$ rather than discarded, so they remain available as negative trajectories for policy optimization.

The rollout process continues until no active branch remains. The final rollout group is constructed from all completed trajectories.

$$\mathcal{G}(x) = \mathcal{C}. \tag{6}$$

Importantly, process scores are used only to decide where to allocate additional rollout budget; they do not replace the task reward. The resulting group contains both successful and failed trajectories generated from shared intermediate states, providing diverse samples for GRPO-style relative advantage estimation.

**Algorithm 1** PATR Rollout Generation

**Require:** Task prompt $x$; policy $\pi_\theta$; process scorer $S_\phi$; budgets $(B_0, K, M)$; thresholds $(\alpha, \Theta)$
Initialize $\mathcal{A}$ with $B_0$ branches from $x$ under $\pi_\theta$; $\mathcal{C} \leftarrow \emptyset$
**while** $\mathcal{A} \neq \emptyset$ **do**
  Roll out each branch in $\mathcal{A}$ for up to $K$ steps under $\pi_\theta$
  Move terminated branches from $\mathcal{A}$ to $\mathcal{C}$
  **if** $\mathcal{A} \neq \emptyset$ **then**
    Compute $s_i^t = S_\phi(x, \tau_i^t)$ for each $\tau_i^t \in \mathcal{A}$ Eq. 3
    Select high-scoring expansion set $\mathcal{E} \subseteq \mathcal{A}$
    Let $\mathcal{L} \subseteq \mathcal{A} \setminus \mathcal{E}$ be branches satisfying the low-score or $\Theta$-repeat criterion Eq. 5
    Sample child set $\mathcal{B} = \{\tau_i^{t,m} : \tau_i^t \in \mathcal{E}, m = 1, \ldots, M\}$ Eq. 4
    $\mathcal{A} \leftarrow (\mathcal{A} \setminus (\mathcal{E} \cup \mathcal{L})) \cup \mathcal{B}$; $\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{L}$
  **end if**
**end while**
**return** $\mathcal{G}(x) = \{\tau \in \mathcal{C} : |\tau| > 0\}$

### 4.2 Task-Adaptive Process Scoring

PATR abstracts task-specific intermediate feedback into a scalar process score as defined in Eq. 3, This interface supports different scorer instantiations, including domain-specific heuristics, pretrained PRMs, and LLM judges, without changing the rollout algorithm or the GRPO objective.

**Heuristic scorer.** When intermediate progress is directly observable, we instantiate $S_\phi$ as a lightweight heuristic over the current state and accumulated step rewards. For example, in navigation-style tasks, the scorer can combine the reward accumulated so far with a normalized progress measure such as distance to the goal. This instantiation requires no learned model and provides a controlled setting for studying adaptive tree rollout when reliable progress signals are available.

**Pretrained PRM scorer.** For tasks where a pretrained process reward model is available, we convert each partial trajectory into a problem–response format. The problem field contains the task prompt, while the response field serializes recent interaction steps, including agent actions, observations, and available step rewards. The PRM produces step-level scores $\{c_\ell\}_{\ell=1}^{L_i}$ at designated step boundaries, which are aggregated into a branch score:

$$s_i^t = f_{\text{agg}}(c_1, \ldots, c_{L_i}), \tag{7}$$

where $f_{\text{agg}}$ can be the last-step score, the mean score, the maximum score, or a weighted combination of the last and mean scores. For long trajectories, we truncate the serialized input while preserving the task prompt and recent interaction steps, which are most relevant for deciding whether to expand the current branch.

**LLM-as-judge scorer.** For complex agentic tasks where no reliable hand-crafted signal or pretrained PRM is sufficient, we instantiate $S_\phi$ with an LLM judge. The judge receives the task prompt and a compact representation of the partial trajectory, including recent actions, observations, and automatically extracted trajectory signals. Specifically, to make scoring sensitive to different stages of SWE-style problem solving, we use phase-specific judge prompts for diagnosis, editing, and verification. The judge is instructed to output a normalized JSON score indicating whether continuing from the current state is likely to lead to a successful solution. We further blend the judge output with a lightweight heuristic based on trajectory features such as repeated actions, error or test signals in observations, evidence before editing, and targeted verification:

$$s_i^t = \lambda s_{\text{judge}}(\tau_i^t) + (1-\lambda) s_{\text{heur}}(\tau_i^t). \quad (8)$$

This design provides robust process scores for long-horizon tool-use tasks while keeping the interface to PATR identical to the heuristic and pretrained-PRM cases. Appendix B provides the full prompts and describes the heuristic scoring signals in detail.

### 4.3 GRPO Training with Tree-Generated Rollout Groups

After tree rollout construction, PATR uses $\mathcal{G}(x)$ from Eq. 6 as the rollout group for GRPO. For each trajectory $\tau_i \in \mathcal{G}(x)$, we compute its outcome reward $r_i^{\text{out}} = r^{\text{out}}(\tau_i)$ and normalize rewards within the tree-generated group,

$$A_i^{\text{tree}} = \frac{r_i^{\text{out}} - \mu_{\mathcal{G}}(x)}{\sigma_{\mathcal{G}}(x) + \varepsilon}, \quad (9)$$

where $\mu_{\mathcal{G}}(x)$ and $\sigma_{\mathcal{G}}(x)$ are computed over all valid trajectories in $\mathcal{G}(x)$. The policy is then updated with the same clipped GRPO objective in Eq. 2, replacing the independently sampled rollout group with the tree-generated group.

**Remark**. PATR departs from the uniform independent sampling used in vanilla GRPO by adaptively allocating more continuations to selected partial histories. This introduces a controlled and localized bias in the rollout group. Importantly, the process scorer is used only for branch selection: it does not alter the policy model, task reward, advantage normalization, or optimization objective. After tree construction, completed, failed, and early-stopped branches are all retained as outcome-labeled trajectories, and the policy is updated using standard GRPO with task rewards. This idea coincides with the efficiency-bias trade-off discussion in the literature of reinforcement learning, where non-uniform allocation of computation or samples can improve data efficiency by focusing efforts on more informative states or transitions (Kocsis and Szepesvári, 2006; Schaul et al., 2016; Espeholt et al., 2018). In PATR, the induced mismatch is further limited because actions are still generated by the current rollout policy conditioned on realized histories, and only the number of continuations allocated to each partial history is adapted. Thus, PATR trades strict uniform rollout sampling for more informative rollout groups while keeping the learning objective unchanged.

## 5 Experiments

### 5.1 Experimental Setup

**Tasks, datasets, and models.** We evaluate PATR on two settings with different levels of interaction complexity. **FrozenLake** (Brockman et al., 2016) is a grid-world navigation task in which an agent must reach a goal while avoiding holes, providing a controlled testbed for multi-turn exploration. We use the rLLM FrozenLake environment (Tan et al., 2025) with procedurally generated task prompts. **SWE-agent training** targets long-horizon software-engineering tasks, where agents must inspect repositories, edit code, and verify fixes through tool interaction. For training, we use a filtered subset of R2E-Gym-Lite-with-Difficulty, which is derived from R2E-Gym (Jain et al., 2025) by removing the most difficult instances. We evaluate on SWE-Bench Verified (Jimenez et al., 2024), a human-filtered set of 500 instances designed for reliable evaluation of coding agents and language models. For FrozenLake, we train Qwen2.5-0.5B-Instruct and Qwen2.5-3B-Instruct (Yang et al., 2024); for SWE-agent training, we use Qwen3-4B-Instruct-2507 (Yang et al., 2025).

**Baselines.** We compare against representative GRPO-based and tree-style rollout strategies under the same policy optimization objective. **GRPO** (Shao et al., 2024) samples independent complete trajectories for each task. **DAPO** (Yu et al., 2026) strengthens GRPO with token-level policy-gradient refinements, rejection sampling, and asymmetric clipping. **ARPO** (Dong et al., 2025b) performs adaptive branching for agentic rollouts based on uncertainty signals rather than process-level trajectory quality. **Tree-Random**

uses the same tree rollout structure as PATR but replaces process-guided selection with random branch allocation.

**Evaluation.** For FrozenLake, we follow the standard success-rate evaluation. Each test prompt is sampled four times, and we report the average fraction of successful task instances. For SWE-Bench evaluation, we use the SWE-agent scaffold with Docker-based execution environments. Each trained agent generates one trajectory per SWE-Bench Verified instance, and we report the resolved rate, following the standard SWE-Bench protocol.

**Implementation Details.** All experiments are implemented with the rLLM framework (Tan et al., 2025). Across both tasks, PATR starts from four initial branches and samples two child continuations when expanding a branch. Branching is performed at fixed intervals, with $K = 5$ for FrozenLake and $K = 13$ for SWE-agent training. We set the maximum interaction length to 10 turns for FrozenLake and 50 turns for SWE-agent training, with maximum response lengths of 10K and 32K tokens, respectively. Early stopping uses the score-margin rule in Eq. 5 with $\alpha = 0.5$, together with an action-loop criterion that terminates branches repeating the same action six times.

The process scorer is instantiated according to the task. For FrozenLake, we use a lightweight progress heuristic based on accumulated rewards and shortest-path distance to the goal, treating holes as blocked cells. For SWE-agent training, we evaluate two scorer variants: **PATR-PRM**, which uses Skywork-o1-Open-PRM-Qwen-2.5-7B (He et al., 2025), and **PATR-Judge**, which uses Qwen2.5-Coder-7B-Instruct (Hui et al., 2024) as an LLM judge with phase-specific prompts. Both scorers operate on recent trajectory context; additional scoring details are provided in Appendix A. To ensure a fair comparison, GRPO and DAPO use a rollout group size of 8, while ARPO, Tree-Random, and PATR use the same number of initial branches. Detailed training hyperparameters and hardware settings are reported in Appendix A.

### 5.2 Main Results

**FrozenLake.** We report the final FrozenLake performance across two model scales in Table 1. PATR achieves the highest success rate and Pass@4 for both Qwen2.5-0.5B-Instruct and Qwen2.5-3B-Instruct. The gain is especially pronounced for the smaller model, where PATR improves the success rate from $66.5\%$ with GRPO to $75.8\%$, suggesting that process-guided rollout allocation is particularly useful when the base policy is less reliable. Beyond accuracy, PATR also produces the shortest average response length on both model scales, indicating that the gains do not come from simply exploring longer trajectories. The comparison with Tree-Random further isolates the effect of process guidance. Although Tree-Random benefits from shared-prefix rollout construction and is competitive in some settings, it remains below PATR in both success rate and Pass@4. This shows that PATR gains from combining tree-structured exploration with quality-aware expansion, rather than from tree construction alone. DAPO also improves over GRPO but often produces longer trajectories, highlighting the difference between objective-level improvements and rollout-level allocation.

| Method | Succ. Rate ↑ | Pass@4 ↑ | RLen ↓ | Turns ↓ |
|---|---|---|---|---|
| ***Qwen2.5-0.5B-Instruct*** | | | | |
| GRPO | 66.5 | 67.0 | 513 | 3.4 |
| DAPO | 71.0 | 71.0 | 562 | 4.2 |
| ARPO | 69.7 | 72.0 | 377 | **3.3** |
| Random | 56.3 | 63.0 | 400 | 3.9 |
| **PATR (ours)** | **75.8** | **78.0** | **344** | 3.6 |
| ***Qwen2.5-3B-Instruct*** | | | | |
| GRPO | 66.8 | 71.0 | 367 | **2.6** |
| DAPO | 72.7 | 73.0 | 533 | 2.7 |
| ARPO | 68.7 | 72.0 | 497 | 3.5 |
| Tree-Random | 72.7 | 75.0 | 419 | 3.6 |
| **PATR (ours)** | **74.3** | **79.0** | **327** | 2.8 |

Table 1: FrozenLake results across two model scales. We report success rate, Pass@4, and average response length (RLen) for each rollout method.

| Method | Resolved Rate ↑ | Env Done ↑ | Turns ↓ |
|---|---|---|---|
| GRPO | 22.2 | 79.2 | 25.8 |
| DAPO | 24.6 | 71.4 | 31.6 |
| ARPO | 25.4 | 74.0 | 27.3 |
| Tree-Random | 24.8 | 84.4 | 30.1 |
| **PATR-Judge (ours)** | 26.0 | 90.8 | **24.1** |
| **PATR-PRM (ours)** | **27.2** | **92.2** | 24.4 |

Table 2: SWE-Bench results with Qwen3-4B-Instruct-2507. We compare methods using resolved rate, environment completion rate (Env Done), and average turns.

**SWE-Bench.** We next evaluate the more challenging SWE-Bench Verified setting in Table 2. Both PATR variants outperform all baselines in the resolved rate. PATR-PRM achieves the best result, improving over GRPO by $+5.0$ points and over the strongest baseline ARPO by $+1.8$ points. PATR-Judge also improves over all baselines, showing that the framework can benefit from either pre-trained PRM scoring or LLM-as-judge feedback.

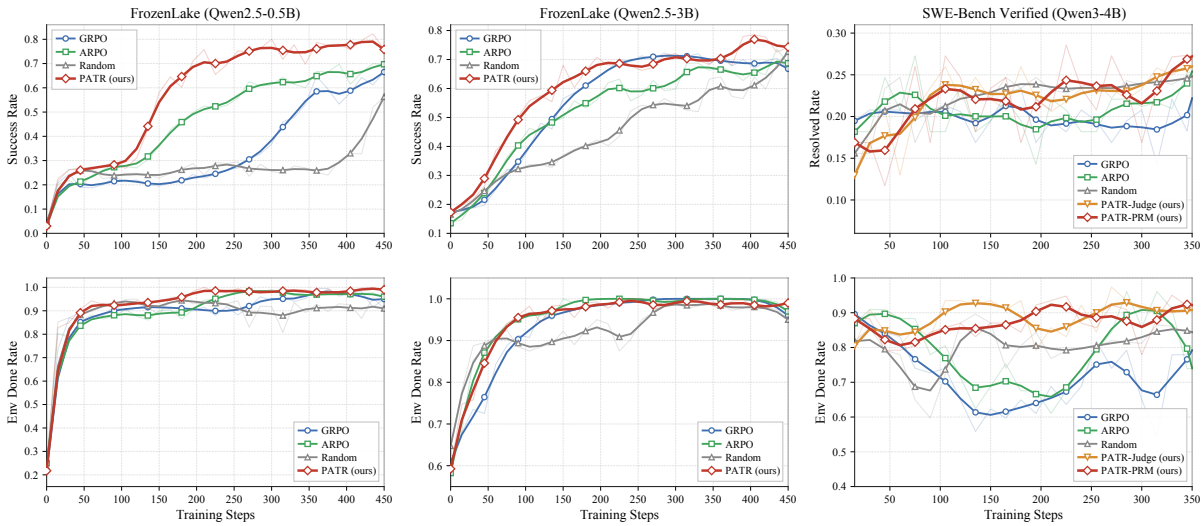


Figure 2: Training curves on FrozenLake and SWE-Bench across methods. PATR improves performance earlier in training and maintains stronger trajectory-completion behavior throughout optimization.

Both PATR-PRM and PATR-Judge complete substantially more task instances than the baselines while using fewer turns on average than DAPO, ARPO, and Tree-Random. This suggests that process-guided expansion helps the policy learn more directed interaction patterns that reach natural task completion. Tree-Random achieves a high environment-done rate but lower resolved rate, indicating that tree rollouts alone can improve completion behavior, while process-guided branch selection is needed to better align exploration with successful issue resolution. The stronger performance of PATR-PRM suggests that step-level PRM scores provide more stable branch-ranking signals for SWE-style code-editing tasks, while the judge variant remains a flexible alternative when task-specific PRMs are unavailable.

### 5.3 Training Dynamics

We further analyze training dynamics in Figure 2. PATR reaches higher success rates earlier than the baselines on FrozenLake, especially with the smaller Qwen2.5-0.5B model. This supports the hypothesis that process-guided branching improves sample efficiency by focusing exploration around partial trajectories that already show progress. The larger gap in the low-capacity setting further suggests that adaptive rollout construction can compensate for weaker initial policies by providing more informative training groups.

On SWE-Bench, PATR-PRM and PATR-Judge maintain higher resolved rates and environment-done rates across training. In contrast, flat rollout methods improve more slowly or plateau earlier, while tree-based baselines without process-guided selection show less consistent gains. The training curves therefore reinforce the main results: shared-prefix tree construction improves rollout diversity, but quality-aware expansion is important for converting additional exploration into reliable task progress. Overall, the dynamics indicate that PATR improves not only final performance but also the efficiency and stability of multi-turn agent RL.

## 6 Analysis

### 6.1 Effect of Tree Hyperparameters

We examine how tree construction choices affect PATR-PRM on SWE-Bench Verified. As shown in Table 3, the default setting, with branch factor $M{=}2$, scoring interval $K{=}13$, and top-$k{=}2$, achieves the best resolved rate. Expanding more branches at each checkpoint slightly reduces performance, suggesting that overly broad expansion weakens the process-score filter. Increasing the branch factor to $M{=}3$ also hurts performance, especially when expansion is concentrated on a single selected branch, indicating reduced rollout-group diversity. Longer scoring intervals further degrade performance, showing that delayed rescoring limits the ability to redirect exploration during long-horizon interaction. Overall, PATR benefits from a moderate branching strategy that scores frequently while keeping expansion selective.

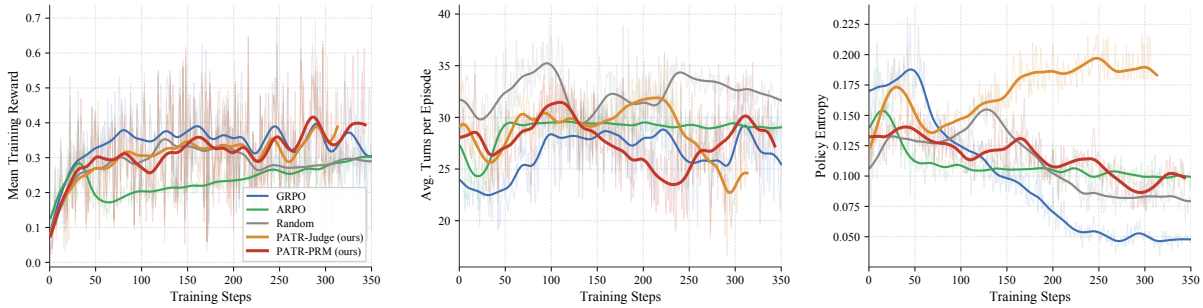


Figure 3: Training dynamics on SWE-Bench. PATR improves the balance between reward acquisition, interaction length, and policy entropy compared with flat and unguided tree-rollout baselines.

| Branch Factor ($M$) | Interval ($K$) | Top-$k$ | Resolved Rate ↑ |
|---|---|---|---|
| *Varying Top-k* | | | |
| 2 | 13 | 4 | 26.6 |
| *Varying Branch Factor* | | | |
| 3 | 13 | 2 | 25.8 |
| 3 | 13 | 1 | 22.0 |
| *Varying Branch Interval* | | | |
| 2 | 15 | 2 | 23.4 |
| 2 | 20 | 2 | 24.6 |
| **PATR-*PRM (default)*** | | | |
| **2** | **13** | **2** | **27.2** |

Table 3: Ablation of tree rollout hyperparameters on SWE-Bench Verified. We vary the branch factor $M$, scoring interval $K$, and number of expanded branches $k$ per checkpoint.

| Method | Rollout Group | Resolved Rate ↑ | Turns ↓ |
|---|---|---|---|
| PATR-PRM | Best_n | 25.8 | 26.4 |
| | Keep All | **27.2** | **24.4** |
| PATR-Judge | Best_n | 23.4 | 31.2 |
| | Keep All | **26.0** | **24.1** |

Table 4: Effect of rollout group construction on SWE-Bench Verified. *Keep All* retains pruned trajectories in the GRPO group, while *Best_n* excludes them.

### 6.2 Effect of Rollout Group Construction

We study whether pruned trajectories should be retained in the rollout group for GRPO training. Table 4 compares *Best_n*, which excludes pruned trajectories, with *Keep All*, which retains them together with completed branches. Keeping all trajectories consistently improves both PATR-PRM and PATR-Judge, with a larger gain for PATR-Judge from 23.4% to 26.0% resolved rate. This supports our design choice of preserving the full tree output for GRPO. Because GRPO computes advantages relative to trajectories from the same task, failed and early-pruned branches can provide useful reward contrast rather than being discarded. The lower average turns under *Keep All* further suggests that pruned trajectories serve as informative negative examples without requiring full interaction budgets.

### 6.3 Training Signal and Policy Dynamics

Figure 3 compares training reward, interaction length, and policy entropy on SWE-Bench. Both PATR variants obtain competitive training rewards while avoiding the excessive interaction length observed in some baselines. PATR-PRM maintains strong reward trends with relatively short episodes, consistent with its higher resolved rate in Table 2. The entropy curves show different exploration patterns: flat GRPO rapidly loses entropy, while PATR-Judge maintains higher entropy for longer and PATR-PRM follows a more conservative profile. These dynamics suggest that process-guided tree rollout improves the balance between exploration, trajectory length, and reward acquisition during training.

## 7 Conclusion

We introduced PATR, a process-guided adaptive tree rollout framework for multi-turn agent RL. Instead of sampling complete trajectories independently, PATR organizes rollout groups as trees and uses task-adaptive process feedback to decide where to expand, preserve, or prune partial trajectories. The resulting rollout groups remain compatible with standard GRPO, since process scores guide rollout construction while policy optimization still relies on task outcome rewards. Experiments on FrozenLake and SWE-Bench show consistent gains over independent-rollout and other rollout-generation baselines, with improvements of up to $+5.0$ points on SWE-Bench and $+9.3$ points on FrozenLake. These results highlight process-guided tree rollouts as a promising direction for effective and scalable multi-turn agent learning.

## Limitations

This work has several limitations. First, PATR relies on the quality of the process scorer used to rank partial trajectories. Although we instantiate the scorer with heuristics, pretrained PRMs, and LLM-as-judge feedback, inaccurate process scores may lead the tree rollout to expand suboptimal branches or prune useful ones. In particular, the pretrained PRM used in our SWE-Bench experiments, Skywork-o1-Open-PRM-Qwen-2.5-7B, may favor actions that generate longer responses, which can affect branch ranking and rollout allocation. Second, our experiments focus on FrozenLake and SWE-Bench. While these benchmarks cover both controlled navigation and realistic long-horizon coding-agent tasks, further evaluation is needed on broader multi-turn agent domains, such as web navigation, embodied interaction, and open-ended tool use.

**LLM Usage Disclosure** LLMs were used only for grammar correction and writing polishing.

## References


Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. Openai gym. *arXiv preprint arXiv:1606.01540*.

Ruike Cao, Shaojie Bai, Fugen Yao, Liang Dong, Jian Xu, and Li Xiao. 2026. Atpo: Adaptive tree policy optimization for multi-turn medical dialogue. *arXiv preprint arXiv:2603.02216*.

Ganqu Cui, Lifan Yuan, Zefan Wang, Hanbin Wang, Yuchen Zhang, Jiacheng Chen, Wendi Li, Bingxiang He, Yuchen Fan, Tianyu Yu, and 1 others. 2025. Process reinforcement through implicit rewards. *arXiv preprint arXiv:2502.01456*.

Aladin Djuhera, Swanand Ravindra Kadhe, Farhan Ahmed, Heiko Ludwig, and Holger Boche. 2026. Tsr: Trajectory-search rollouts for multi-turn rl of llm agents. *arXiv preprint arXiv:2602.11767*.

Guanting Dong, Licheng Bao, Zhongyuan Wang, Kangzhi Zhao, Xiaoxi Li, Jiajie Jin, Jinghan Yang, Hangyu Mao, Fuzheng Zhang, Kun Gai, and 1 others. 2025a. Agentic entropy-balanced policy optimization. *arXiv preprint arXiv:2510.14545*.

Guanting Dong, Hangyu Mao, Kai Ma, Licheng Bao, Yifei Chen, Zhongyuan Wang, Zhongxia Chen, Jiazhen Du, Huiyang Wang, Fuzheng Zhang, and 1 others. 2025b. Agentic reinforced policy optimization. *arXiv preprint arXiv:2507.19849*.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. 2018. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. *Preprint*, arXiv:1802.01561.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. 2026. Group-in-group policy optimization for llm agent training. *Advances in Neural Information Processing Systems*, 38:46375–46408.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. 2023. Reasoning with language model is planning with world model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 8154–8173.

Jujie He, Jiacai Liu, Chris Yuhao Liu, Rui Yan, Chaojie Wang, Peng Cheng, Xiaoyu Zhang, Fuxiang Zhang, Jiacheng Xu, Wei Shen, and 1 others. 2025. Skywork open reasoner 1 technical report. *arXiv preprint arXiv:2505.22312*.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, and 1 others. 2024. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*.

Naman Jain, Jaskirat Singh, Manish Shetty, Liang Zheng, Koushik Sen, and Ion Stoica. 2025. R2e-gym: Procedural environments and hybrid verifiers for scaling open-weights swe agents. *arXiv preprint arXiv:2504.07164*.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, volume 2024, pages 54107–54157.

Levente Kocsis and Csaba Szepesvári. 2006. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer.

Jing Yu Koh, Stephen McAleer, Daniel Fried, and Ruslan Salakhutdinov. 2024. Tree search for language model agents. *arXiv preprint arXiv:2407.01476*.

Jiazheng Li, Yawei Wang, Qiaojing Yan, Yijun Tian, Zhichao Xu, Huan Song, Panpan Xu, and Lin Lee Cheong. 2026a. Salt: Step-level advantage assignment for long-horizon agents via trajectory graph. In *Findings of the Association for Computational Linguistics: EACL 2026*, pages 4709–4725.

Weijie Li, Jin Wang, Liang-Chih Yu, and Xuejie Zhang. 2026b. Step-grpo: Enhancing reasoning quality and efficiency via structured prm-based reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 31734–31742.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. Let's verify step by step. In *International Conference on Learning Representations*, volume 2024, pages 39578–39601.

Xiaoqian Liu, Ke Wang, Yuchuan Wu, Fei Huang, Yongbin Li, Junge Zhang, and Jianbin Jiao. 2025. Agentic reinforcement learning with implicit step rewards. *arXiv preprint arXiv:2509.19199*.

Gagan Mundada, Zihan Huang, Rohan Surana, Sheldon Yu, Jennifer Yuntong Zhang, Xintong Li, Tong Yu, Lina Yao, Jingbo Shang, Julian McAuley, and 1 others. 2026. Ws-grpo: Weakly-supervised group-relative policy optimization for rollout-efficient reasoning. *arXiv preprint arXiv:2602.17025*.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. 2016. Prioritized experience replay. *Preprint*, arXiv:1511.05952.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652.

Rohan Surana, Gagan Mundada, Xunyi Jiang, Chuhan Wang, Zhenwei Tang, Difan Jiao, Zihan Huang, Yuxin Xiong, Junda Wu, Sheldon Yu, Xintong Li, Raghav Jain, Nikki Kuang, Sizhe Zhou, Bowen Jin, Zhendong Chu, Tong Yu, Ryan Rossi, Kuan-Hao Huang, and 3 others. 2026. Generate, filter, control, replay: A comprehensive survey of rollout strategies for llm reinforcement learning. *Preprint*, arXiv:2605.02913.

Sijun Tan, Michael Luo, Colin Cai, Tarun Venkat, Kyle Montgomery, Aaron Hao, Tianhao Wu, Arnav Balyan, Manan Roongta, Chenguang Wang, and 1 others. 2025. rllm: A framework for post-training language agents.

Hanlin Wang, Chak Tou Leong, Jiashuo Wang, Jian Wang, and Wenjie Li. 2025. Spa-rl: Reinforcing llm agents via stepwise progress attribution. *arXiv preprint arXiv:2505.20732*.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. 2024. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9426–9439.

Shangyu Xing, Siyuan Wang, Chenyuan Yang, Xinyu Dai, and Xiang Ren. 2025. Lookahead tree-based rollouts for enhanced trajectory-level exploration in reinforcement learning with verifiable rewards. *arXiv preprint arXiv:2510.24302*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, and 43 others. 2024. Qwen2 technical report. *Preprint*, arXiv:2407.10671.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, and 1 others. 2026. Dapo: An open-source llm reinforcement learning system at scale. *Advances in Neural Information Processing Systems*, 38:113222–113244.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, and 1 others. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623.

## A Additional Implementation Details

**Training details.** We use the same optimization hyperparameters across methods unless otherwise specified. The learning rate is $1\times10^{-6}$, the clipping ratio is $0.28$, the KL coefficient is $0.001$, and the

rollout sampling temperature is 1.0. FrozenLake is trained with batch size 64 for 400 steps, while SWE-agent training uses batch size 8 for 300 steps. FrozenLake experiments are run on one A100 node, and SWE-agent experiments are run on one H200 node.

**Process scoring details.** For FrozenLake, the process scorer combines accumulated step rewards with a normalized BFS shortest-path distance to the goal, where holes are treated as blocked cells. For SWE-agent training, both PATR-PRM and PATR-Judge score each branch using recent trajectory context. PATR-PRM serializes the most recent 10 interaction steps and feeds them to Skywork-o1-Open-PRM-Qwen-2.5-7B, which returns step-level process scores; we use the score of the most recent step as the branch score. PATR-Judge provides Qwen2.5-Coder-7B-Instruct with the task context and a compact representation of recent actions, observations, tool types, and extracted trajectory signals. The judge outputs a trajectory-level score, which is blended with a lightweight heuristic that rewards useful signals such as targeted testing and evidence-based editing while penalizing repeated actions and premature termination. We describe the heuristic and provide all judge prompts in Appendix B.

**Scoring infrastructure.** For SWE-bench experiments, the PRM and LLM-judge models are hosted on separate vLLM inference servers and queried during expansion checkpoints. This separates policy rollout generation from process scoring and avoids GPU contention during training.

## B LLM-as-Judge Scoring Details

This section provides additional details about the LLM-as-judge scorer used in PATR-Judge for SWE-Bench. At each scoring checkpoint, the judge receives the task description and a compact representation of the current partial trajectory, including recent agent actions, environment observations, inferred tool types, and automatically extracted trajectory signals. The judge outputs a scalar score in $[0, 1]$ indicating how promising the current branch is for further expansion. To make the evaluation more stage-aware, we use different prompts for diagnosis, editing, and verification phases.

**Phase selection.** The prompt is selected automatically from recent interaction patterns. If the recent trajectory contains test-related actions or observations, such as running `pytest`, `unittest`, or observing pass/fail messages, we use the verification prompt. If the recent trajectory contains file modification actions, such as patching, insertion, or replacement, we use the editing prompt. If the recent trajectory mainly contains repository exploration, such as search, file inspection, traceback analysis, or command-line reading, we use the diagnosis prompt. When no specific phase is detected, we use the default prompt. This design allows the judge to evaluate the type of progress that is most relevant to the current stage of problem solving.

**Heuristic score.** In addition to the LLM score, we compute a lightweight heuristic score from recent trajectory signals. The heuristic starts from a neutral value and adjusts the score according to observable behaviors. It penalizes repeated identical actions, premature finish actions, command failures, and edits made without prior evidence from search, file inspection, or testing. It rewards signals associated with useful progress, including targeted test execution, passing tests, informative tracebacks, code or file information discovered during exploration, and edits made after relevant evidence has been collected. The final branch score is a weighted combination of the LLM judgment and the heuristic score, with the LLM judgment receiving the larger weight. The heuristic acts as a stabilizing signal, especially when the LLM judge assigns overly optimistic scores to repetitive or weakly grounded trajectories.

**Judge prompts.** We list the system prompt and all phase-specific scoring prompts below 1, 2, 3, 4, and 5. The placeholders `{task}` and `{trajectory}` are filled with the task description and the compact trajectory representation at each scoring checkpoint.

```
You evaluate partial SWE-agent trajectories for tree-search expansion.
Output ONLY valid JSON: {"score": <0.0 to 1.0>}. Nothing else.
```

Code 1: System prompt for the LLM-as-judge scorer.

```
Score this partial SWE-agent trajectory for tree-search branching.

Consider:
- Is the agent making measurable progress toward solving the issue?
- Are its actions grounded in evidence from the code or observations?
- Is it avoiding repetitive or aimless commands?
- If we continue from this state, how likely will this branch lead to a correct fix?

0.0 = stuck, looping, or wrong direction
0.5 = some relevant work but unclear progress
1.0 = clearly on track, strong evidence of progress

Task:
{task}

Trajectory:
{trajectory}

Output ONLY: {"score": <0.0 to 1.0>}
```

Code 2: Default judge prompt.

```
Score this partial trajectory during EXPLORATION / DIAGNOSIS.

Consider:
- Is the agent searching in the right files and directories?
- Is it narrowing down the bug location with each step?
- Is it gathering evidence (reading code, reproducing the bug) rather than guessing?
- Is it avoiding repeated searches that yield no new information?

0.0 = aimless browsing, wrong files, repetitive commands
0.5 = relevant exploration but slow progress
1.0 = efficiently locating the root cause

Task:
{task}

Trajectory:
{trajectory}

Output ONLY: {"score": <0.0 to 1.0>}
```

Code 3: Judge prompt for the diagnosis phase.

```
Score this partial trajectory during EDITING.

Consider:
- Is the agent editing the file that evidence pointed to?
- Is the change minimal and tied to the diagnosed root cause?
- Did the agent collect enough evidence before editing, or is it guessing?
- Could this edit introduce new bugs or break other functionality?

0.0 = editing blind, wrong file, speculative changes
0.5 = reasonable edit but unclear if it addresses root cause
1.0 = precise, evidence-based fix in the right location

Task:
{task}

Trajectory:
{trajectory}

Output ONLY: {"score": <0.0 to 1.0>}
```

Code 4: Judge prompt for the editing phase.

```
Score this partial trajectory during TESTING / VERIFICATION.

Consider:
- Is the agent running tests relevant to the issue (not random test suites)?
- If tests fail, does the agent appear to understand the failure?
- If tests pass, do they actually cover the bug that was fixed?
- Is the agent verifying the fix rather than declaring premature success?

0.0 = irrelevant tests, ignoring failures, false success
0.5 = running some tests but coverage of the fix is unclear
1.0 = targeted verification that confirms the fix works

Task:
{task}

Trajectory:
{trajectory}

Output ONLY: {"score": <0.0 to 1.0>}
```

Code 5: Judge prompt for the verification phase.