REWARD-GUIDED TREE SEARCH FOR EFFICIENT AGENT TEST-TIME SCALING

Anonymous authors
Paper under double-blind review

ABSTRACT

Test-time scaling improves the performance of large language model agents by allocating additional computation during inference. However, independent best-of-N sampling is costly on long-horizon tasks, as each new attempt repeats reasoning and environment interactions that could be shared across trajectories. Recent approaches address this inefficiency through trajectory reuse or intermediate guidance, but reuse based on completed attempts postpones branching until those attempts finish, while costly or unreliable intermediate evaluations can offset computational savings or misdirect search. We introduce SPAR, a reward-guided tree search framework that combines adaptive computation allocation with prefix reuse for efficient agent test-time scaling. During generation, a compact reward model periodically ranks active prefixes, guiding SPAR to branch from promising states and prune persistently low-ranked branches to reallocate available computation. Each new branch inherits its parent's interaction history and execution state, allowing alternative continuations to be explored without regenerating or re-executing their shared prefix. To support these decisions, the reward model is jointly trained with a confidence-weighted pairwise objective for comparing prefixes and candidate actions using sampled continuation success rates, and a listwise objective for selecting completed trajectories using verified outcomes, connecting search and selection within a single framework. Compared with best-of-16 sampling, SPAR achieves higher accuracy with 46.3% fewer generated tokens on SWE-bench Verified and 58.4% fewer on AIME 2024, where accuracy improves by 3.3 percentage points.

1 INTRODUCTION

Test-time scaling (TTS) improves large language model (LLM) performance by allocating additional computation during inference. This computation can support longer reasoning and iterative refinement (Muennighoff et al., 2025; Madaan et al., 2023), multiple candidate solutions followed by aggregation or verification (Wang et al., 2022; Lightman et al., 2024), and structured search over intermediate states (Yao et al., 2023). The resulting gains depend on how the inference budget is distributed across these strategies (Snell et al., 2024). For tool-using agents, additional computation can increase the number of sampled trajectories, extend interaction within each attempt, or evaluate more alternatives at individual decision points (Ehrlich et al., 2025; Zhu et al., 2025; Shen et al., 2025). Agent TTS therefore scales both reasoning and environment interaction, creating opportunities to explore and revise solutions but also increasing the cost of repeated attempts.

This cost becomes particularly pronounced on long-horizon tasks such as repository-level software engineering. An agent may inspect many files, execute commands, modify code, and run tests before producing a candidate solution. Independent best-of-N (BoN) sampling repeats a complete trajectory for each attempt, even when alternative solutions could build on the same earlier progress. For example, different repairs may share repository exploration and fault localization before diverging at a patch decision. Completing every attempt also commits further computation to paths that intermediate feedback could have deprioritized. Finally, producing a successful candidate does not ensure that it will be selected (Li et al., 2026b), leaving a verification gap between candidate coverage and realized task success. Efficient agent TTS therefore requires economical exploration together with reliable final selection.

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

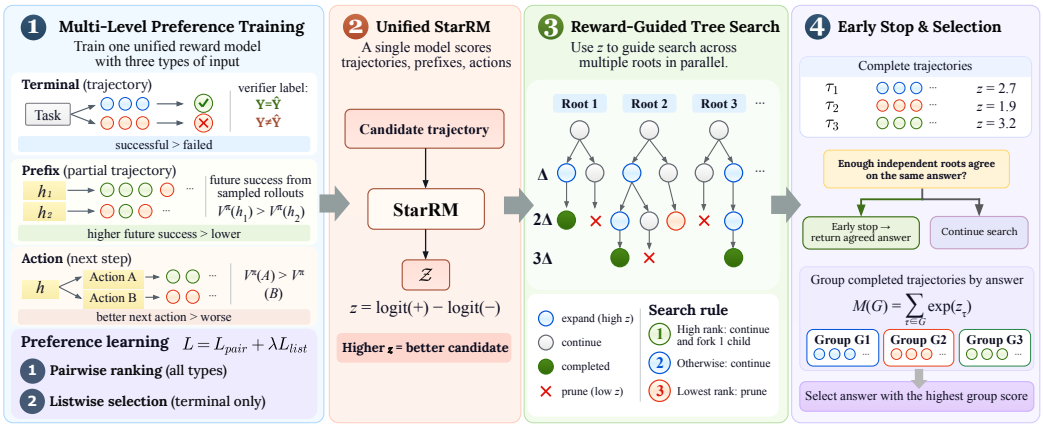


Figure 1: Overview of SPAR. SPAR reuses trajectory prefixes and reallocates available computation to promising continuations through reward-guided branching and pruning, improving the accuracy and efficiency of agent test-time scaling.

Recent approaches improve on independent sampling through trajectory reuse and intermediate guidance. SWE-Replay resumes from selected prefixes of completed attempts, reducing redundant execution but delaying alternative continuations until the initial attempt finishes (Ding & Zhang, 2026). Other methods guide exploration during generation: SWE-Search evaluates states within tree search (Antoniades et al., 2025), AgentRM and QLASS learn signals for stepwise search (Xia et al., 2025; Lin et al., 2025), and SWE-TRACE uses rubric-conditioned action evaluation (Han et al., 2026). Uncertainty-based allocation further concentrates sampling at contentious decisions (Lee et al., 2026b). These works reduce waste, yet effective allocation across unfinished trajectories remains challenging. Prefix reuse alone lowers continuation cost without identifying which prefixes merit further expansion, while choosing the next action within one trajectory does not determine how to distribute a shared budget across competing trajectories. The required comparisons must assess the prospects of partial solutions before their outcomes are known, so ranking errors can repeatedly direct computation toward unproductive paths. Generative evaluations also add decoding overhead at each decision point (Antoniades et al., 2025; Lee et al., 2026a). The central challenge is therefore to combine prefix reuse with timely allocation across active trajectories, using feedback that is reliable enough to guide exploration and inexpensive enough to preserve its savings.

We introduce SPAR, a reward-guided tree search framework that combines adaptive computation allocation with prefix reuse for efficient long-horizon agent TTS (Figure 1). It allocates additional computation to promising trajectory prefixes and explores alternative continuations without repeating the reasoning and environment interactions. Starting from several independent attempts, the framework uses a compact reward model to periodically compare active prefixes, branch from promising states, and prune persistently low-ranked branches to release capacity. New branches inherit their parents' interaction histories and execution states, exploring alternative continuations without regenerating or re-executing shared prefixes. Expansion proceeds in small batches, allowing later evaluations to revise allocation while independent starting points preserve diversity. The same reward model supports final solution selection, connecting decisions about where to generate candidates with the choice of which solution to return.

To support these decisions, SPAR incorporates StarRM, a compact evaluator trained with relative ranking objectives for trajectory prefixes, candidate actions, and completed trajectories. Verified outcomes establish preferences between completed trajectories. For prefixes and candidate actions, preferences are estimated from the success rates of multiple sampled continuations. A pairwise objective learns these comparisons across all three input types, with confidence weights reducing the influence of uncertain preferences inferred from finite continuation samples. A jointly optimized listwise objective rewards assigning selection probability to successful solutions within sets of completed trajectories. At inference time, the evaluator directly produces scores without generating evaluation rationales. These learned comparisons enable SPAR to direct branching and pruning during

generation and select among completed solutions, integrating reward modeling into the framework's allocation and selection decisions.

We evaluate test-time scaling and trajectory scoring on software engineering and mathematics. SPAR outperforms BoN@16 using 46.3% fewer generated tokens on SWE-bench Verified and 58.4% fewer on AIME 2024. StarRM-9B outperforms all evaluated SWE reward models and remains competitive with strong GPT-based judges across domains and generation models.

Our contributions are threefold:

- We introduce SPAR, which allocates computation across active trajectories through selective branching and pruning while reusing executed prefixes and preserving alternative search directions.
- We train StarRM, a compact reward model that jointly learns pairwise and listwise preferences to support intermediate allocation and final solution selection.
- We demonstrate efficient test-time scaling on software engineering and mathematics, with a competitive reward model for trajectory ranking.

2 Related Work

Test-time scaling. Test-time scaling improves inference through sampling, verification, and adaptive computation allocation (Wang et al., 2022; Cobbe et al., 2021; Snell et al., 2024). Sequential approaches, such as Self-Refine, Reflexion, and Satori-SWE (Madaan et al., 2023; Shinn et al., 2023; Zeng et al., 2025), improve solutions by iteratively refining existing outputs or incorporating feedback from previous attempts. Complementary approaches explore partial solutions through tree search and intermediate evaluation (Yao et al., 2023; Zhou et al., 2023; Antoniades et al., 2025). For efficient agent inference, SWE-Replay reuses archived prefixes, while CATTS and EGSS adapt sampling using uncertainty or entropy (Ding & Zhang, 2026; Lee et al., 2026b; Mao et al., 2026). Our framework combines selective allocation across active prefixes with execution-state reuse and reward-guided final selection.

Process and outcome reward modeling. Outcome verifiers select completed solutions in mathematics and software engineering (Cobbe et al., 2021; Pan et al., 2024; Jain et al., 2025b). Process reward models provide intermediate guidance using human annotations, rollout-based supervision, or learned process preferences (Lightman et al., 2024; Wang et al., 2024; Luo et al., 2024; Guan et al., 2025). Other approaches derive implicit process rewards from outcome labels or generate reasoning before evaluation (Yuan et al., 2024; Zhao et al., 2026). For agents, AgentRM and QLASS learn search guidance, while Web-Shepherd, SWE-TRACE, and PRInTS incorporate checklists, rubrics, or trajectory summarization (Xia et al., 2025; Lin et al., 2025; Chae et al., 2026; Han et al., 2026; Lee et al., 2026a). Our framework jointly trains a compact evaluator with confidence-weighted pairwise and terminal listwise objectives to support intermediate allocation and final selection.

3 Framework

We introduce SPAR, a reward-guided tree search framework for efficient test-time scaling of long-horizon agents. It allocates additional sampling to promising trajectory prefixes and reuses their interaction histories and execution states to reduce the cost of generating new attempts. A compact reward model guides branching and pruning during generation, then scores completed trajectories for final selection. We first describe this adaptive sampling procedure, followed by the reward model and its training objectives.

3.1 Trajectory Generation and Budget

For a task x, a fixed agent policy π generates a response a_t, including reasoning and an action, and receives an environment observation o_t. The trajectory prefix after t turns is $h_t = (x, a_1, o_1, \ldots, a_t, o_t)$. A completed trajectory τ is an attempt that has ended normally or reached an execution limit.

SPAR starts with R independent trajectories, allows at most $L \geq R$ active branches, and collects up to N completed trajectories in a set $\mathcal{C}$. Each path has a turn limit H. A reward model assigns a score $z_\theta(c)$ to a candidate c to guide allocation and final selection. Section 3.4 describes how the model computes these scores and learns from training comparisons.

3.2 Adaptive Rollout Allocation

During trajectory generation, SPAR evaluates active prefixes at task-specific checkpoints, defined by a fixed interval of turns or a designated execution event. Prefixes at the same turn depth are compared together. Allocation depends on which prefixes are more promising within this comparison, so we rank their scores relative to one another. For m active prefixes $h_1, \ldots, h_m$, the rank of prefix i is

$$\rho_i = \frac{1}{m} \sum_{j=1}^{m} \left(\mathbf{1}[z_\theta(h_j) < z_\theta(h_i)] + \tfrac{1}{2} \mathbf{1}[z_\theta(h_j) = z_\theta(h_i)] \right). \tag{1}$$

Higher values indicate more promising prefixes, and ties receive equal ranks. The ranks are recomputed at each checkpoint to identify where additional sampling is most useful as new observations become available.

Stopping persistently weak branches. A low rank at one checkpoint may not reflect a trajectory's eventual outcome. We therefore prune a branch only when it ranks last at two consecutive checkpoints, with at least one strictly higher-scoring branch at each checkpoint. This gives a low-ranked branch another interval to make progress before stopping it and avoids pruning when all scores are tied. Pruned branches are excluded from $\mathcal{C}$ and do not count toward N, freeing capacity for more promising continuations.

Branching from promising prefixes. While $|\mathcal{C}| < N$, SPAR can add branches whenever fewer than L trajectories are active. At each checkpoint, it selects up to two highest-ranked surviving prefixes, limited by the available running slots and number of parents. Each selected parent produces one child, which inherits its interaction history, execution state, and remaining turn budget and samples a new continuation while the parent continues. This shares the cost of the existing prefix across attempts, while expansion in small batches allows later rankings to guide further allocation. When L branches are active, generation continues without new forks until completion or pruning frees a running slot. Generation stops once $|\mathcal{C}| = N$, no active branch remains, or the early-stopping condition in Section 3.3 is met.

3.3 Final Solution Selection and Early Stopping

The reward model scores completed trajectories in $\mathcal{C}$, and we group their outputs using a task-specific matching rule. For example, mathematical outputs can be grouped by equivalent answers, and software patches by whether they modify the same lines in the same files. Let $\mathcal{G}(\mathcal{C})$ denote the resulting collection of groups, where each group $G \subseteq \mathcal{C}$ contains trajectories with matching outputs. We assign each group the normalized support

$$M(G) = \frac{\sum_{\tau \in G} \exp(z_\theta(\tau))}{\sum_{\tau \in \mathcal{C}} \exp(z_\theta(\tau))}, \qquad G \in \mathcal{G}(\mathcal{C}), \tag{2}$$

and return the highest-scoring trajectory in

$$\widehat{G} = \arg\max_{G \in \mathcal{G}(\mathcal{C})} M(G). \tag{3}$$

Grouping combines evidence from multiple attempts with learned trajectory quality, allowing repeated support for an output to contribute to final selection.

Agreement among completed trajectories can also save computation by ending generation early. We stop the remaining branches when all trajectories in $\mathcal{C}$ belong to one group and cover at least $\max(2, \lceil R/2 \rceil)$ distinct initial roots. Forked branches inherit their parent's root identity, so matching descendants from one root count only once toward this threshold. This avoids further generation once a common output group is supported across enough independent starting points.

3.4 TRAINING THE REWARD MODEL

To support SPAR's allocation and selection decisions, we train StarRM, a compact reward model that evaluates both intermediate candidates and completed trajectories. The model learns relative preferences over intermediate candidates and completed solutions through pairwise and listwise objectives.

3.4.1 UNIFIED INPUTS AND REWARD SCORING

StarRM supports three input types within a single model: a trajectory prefix h, a candidate action with its observation (a, o), and a completed trajectory τ. Each training input contains the task description, the candidate's interaction context, and a task-success question. For a prefix, the context consists of all responses and environment observations up to the current turn. For a candidate action, it contains the parent history, the action, and its resulting observation. For a completed trajectory, it contains the full interaction history up to termination. Intermediate inputs also include the remaining turn budget. Appendix A.2 details the input layouts.

For example, a prefix input ends with: *Will continuing from this history solve the task? Answer + for yes or − for no.* For a completed trajectory, the question instead asks whether the attempt solved the task. Let $r(c)$ denote the full formatted input for candidate c. We obtain the model's scalar reward prediction from the difference between its next-token logits for the two answers:

$$z_\theta(c) = \ell_\theta(+ \mid r(c)) - \ell_\theta(- \mid r(c)), \tag{4}$$

where $\ell_\theta(\cdot \mid r(c))$ denotes the model's next-token logits conditioned on the input. The score $z_\theta(c)$ serves as the reward model's numerical output; the ranking objectives below train it to reflect preferences among candidates across all three input types. At inference, the model uses the same input construction and logit computation without generating evaluation rationales.

3.4.2 COMPARISON SUPERVISION AND JOINT TRAINING

Supervision from task outcomes. Branching and final selection require ranking candidates, so we construct training comparisons within the same task and input type. For completed trajectories, a verifier provides the outcome $Y_x(\tau) \in \{0, 1\}$, establishing a preference for successful attempts over failed ones. For unfinished prefixes and candidate actions, we estimate the expected terminal reward of continuing with policy π:

$$V^\pi(c) = \mathbb{E}_{\tau \sim \pi(\cdot|c)}[Y_x(\tau)]. \tag{5}$$

Here c denotes a prefix or a candidate action in its parent context, and $V^\pi(c)$ is the probability of solving the task by continuing from that candidate within the remaining turn budget. We estimate it by Monte Carlo sampling (Wang et al., 2024; Xia et al., 2025):

$$\widehat{V}^\pi(c_i) = \frac{1}{n_i} \sum_{m=1}^{n_i} Y_x(\tau_i^{(m)}) = \frac{k_i}{n_i}, \tag{6}$$

where n_i is the number of sampled continuations and k_i the number of verified successes. Prefix comparisons use prefixes from the same task, while action comparisons use actions sharing the same parent history, with continuations sampled after executing each action. Thus, completed-trajectory preferences come from verified outcomes, and intermediate preferences come from estimated continuation success.

Confidence-weighted pairwise ranking. Finite continuation samples can leave the ordering of intermediate candidates uncertain. To estimate the probability that one candidate is better than another, we represent uncertainty in each candidate's success probability $q_i = V^\pi(c_i)$ with a Beta posterior:

$$q_i \mid k_i, n_i \sim \mathrm{Beta}(k_i + 1, n_i - k_i + 1). \tag{7}$$

This follows from a uniform prior updated with k_i successes and $n_i - k_i$ failures. Treating the candidate posteriors as independent, we compute the probability that i has a higher success rate than j and use it as the preference target:

$$t_{ij} = \Pr(q_i > q_j \mid k_i, n_i, k_j, n_j), \qquad w_{ij} = |2t_{ij} - 1|. \tag{8}$$

The confidence weight w_{ij} is small when either ordering is plausible and approaches one when the evidence favors one candidate. For a verified successful trajectory i paired with a failed trajectory j, the preference is known and $t_{ij} = w_{ij} = 1$. All three input types use the pairwise loss

$$\ell_{\text{pair}}(i, j) = -w_{ij} \left[t_{ij} \log p_{ij} + (1 - t_{ij}) \log(1 - p_{ij}) \right], \tag{9}$$

where $p_{ij} = \sigma(z_i - z_j)$, $z_i = z_\theta(c_i)$, and σ is the sigmoid function. This trains score differences to reflect preferences while reducing the influence of uncertain intermediate comparisons.

Listwise selection of completed trajectories. We additionally train on sets of completed attempts to support final solution selection. Let $\mathcal{S}$ index completed trajectories for task x and $\mathcal{S}^+ = \{i \in \mathcal{S} : Y_x(\tau_i) = 1\}$ index its successful subset. For sets containing both successes and failures, we minimize

$$\ell_{\text{list}}(\mathcal{S}) = -\log \frac{\sum_{i \in \mathcal{S}^+} \exp(z_i)}{\sum_{j \in \mathcal{S}} \exp(z_j)}, \qquad z_i = z_\theta(\tau_i). \tag{10}$$

This increases the total selection probability of successful solutions without imposing an ordering among them. Pairwise training covers all three input types, while listwise training uses only completed trajectories. We jointly optimize

$$\mathcal{L} = \mathcal{L}_{\text{pair}} + \lambda \mathcal{L}_{\text{list}}, \tag{11}$$

where the two terms average their respective losses and λ controls their relative weight. The resulting reward model supplies SPAR with prefix scores for branching and pruning and completed-trajectory scores for final selection, supporting efficient test-time scaling with a fixed agent policy.

4 Experiments

4.1 Experimental Setup

Benchmarks and models. We evaluate test-time scaling on SWE-bench Verified (Jimenez et al., 2024), a human-filtered benchmark of 500 repository-level issues, and the mathematical reasoning benchmarks MATH-500 (Hendrycks et al., 2021) and AIME 2024 (MAA). We use Qwen3.6-27B (Qwen Team, 2026a) and Qwen2.5-7B ReTool (Feng et al., 2026) as the rollout generation models for SWE and math, respectively, keeping their parameters fixed during TTS. ReTool interleaves reasoning with Python execution, allowing us to study agent TTS on these mathematical problems. To assess reward scoring across tasks and generation models, we additionally evaluate on AIME 2025 and SWE trajectories generated by Qwen3.6-35B-A3B (Qwen Team, 2026b) and the GPT-series models (Achiam et al., 2023).

Baselines. We compare SPAR with strategies that allocate generation at different stages. BoN samples independent complete trajectories and selects the one with the highest reward. Moving selection into generation, beam search (Graves, 2012) retains the highest-scoring prefixes at each depth, while CSO (Li et al., 2026a) greedily chooses among executed candidate actions at each turn. MCTS (Świechowski et al., 2023) balances exploration and exploitation using values backed up from completed rollouts, whereas DVTS (Beeching et al.) expands independent subtrees using intermediate rewards. We also include SWE-Replay (Ding & Zhang, 2026) to evaluate trajectory reuse through heuristic sampling from archived prefixes, followed by test filtering and patch voting.

To assess how well StarRM scores agent trajectories, we compare it with domain-specific reward models: OpenHands-4B, 32B (Wang et al., 2026a), and DeepSWE-14B (Luo et al., 2025a) for SWE, and Qwen2.5-Math-PRM-7B (Zhang et al., 2025), ReasonFlux-PRM-7B (Zou et al., 2026), and GenPRM-7B (Zhao et al., 2026) for math. We also use GPT-5.4 and GPT-5.6-Luna as LLM-as-a-judge baselines to score the same trajectories using the rubric in Appendix A.3, with random selection and majority voting providing additional references.

Evaluation. We report task accuracy and average generated completion tokens per task, including tokens spent on pruned branches. For BoN, we vary $N \in \{1, 2, 4, 8, 16\}$; BoN Oracle reports whether any sampled attempt succeeds. For reward evaluation, we sample 16 trajectories once per task and reuse the same pool across all evaluators. Alongside selection accuracy, we report within-task AUC for completed trajectories and prefixes, measuring how often a randomly selected successful example receives a higher score than an unsuccessful example from the same task.

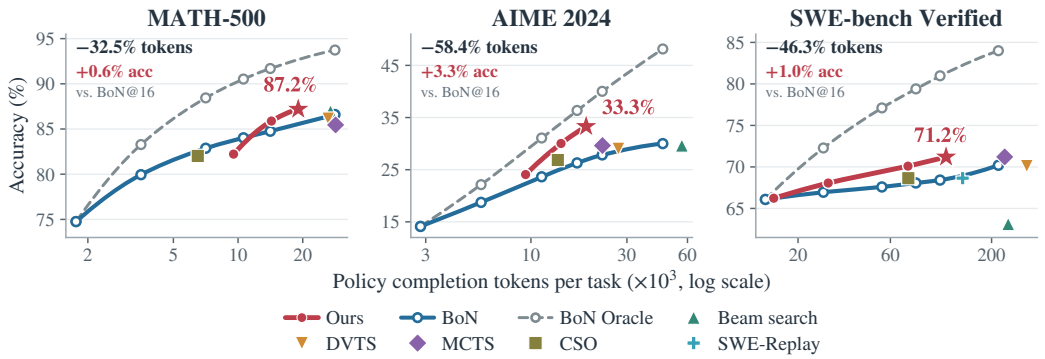


Figure 2: Test-time scaling performance. SPAR improves accuracy over BoN@16 while using fewer generated completion tokens across all three benchmarks. Stars mark SPAR at N = 16; annotated accuracy gains are in percentage points.

Training and implementation. We collect SWE training trajectories from Qwen3.6-27B on SWE-rebench (Badertdinov et al., 2026) and SWE-Gym (Pan et al., 2024) datasets. For mathematics, Qwen2.5-7B ReTool generates trajectories on DeepScaleR-Preview (Luo et al., 2025b). From 77,500 labeled candidates, we construct 97,825 training comparison sets for joint pairwise and listwise supervision, and fine-tune Qwen3.5-9B to obtain StarRM-9B.

During TTS, we allow $L = 12$ concurrent branches and vary the initial trajectory count R and completed-trajectory limit N to measure the accuracy-cost tradeoff. Prefixes are scored every turn for mathematics and every eight turns for SWE, with additional SWE checkpoints after test-execution actions; prefixes at the same depth are ranked together. All experiments, including training and evaluation, run on a single NVIDIA B200 node. Further implementation details are provided in Appendix A.1.

4.2 Test-Time Scaling Performance

Figure 2 compares SPAR with independent sampling and other test-time scaling methods across three benchmarks. With a completed-trajectory budget of $N = 16$ (red stars), SPAR achieves 87.2%, 33.3%, and 71.2% accuracy on MATH-500, AIME 2024, and SWE-bench Verified, respectively. Relative to BoN@16, these results improve accuracy by 0.6, 3.3, and 1.0 percentage points while reducing generated completion tokens by 32.5%, 58.4%, and 46.3%. Thus, allocating the same trajectory budget adaptively yields higher accuracy with substantially less generation.

The advantage extends to methods that already reuse trajectories or guide search. On SWE-bench Verified, SPAR improves over SWE-Replay from 68.6% to 71.2% accuracy while using approximately 17.9% fewer tokens. DVTS approaches our accuracy on MATH-500 and SWE-bench Verified (86.2% and 70.2%), but SPAR still improves both results with approximately 27.5% and 61.7% fewer tokens. It also matches MCTS's 71.2% accuracy on SWE with approximately 50.2% fewer tokens. These results support combining prefix reuse with reward-guided branching, pruning, and final selection. SPAR retains alternative continuations while concentrating computation on promising prefixes, improving the quality of returned solutions without independently regenerating each attempt.

4.3 Reward Model Results

Tables 1 and 2 compare evaluators on the same pools of 16 trajectories per task, measuring final-selection accuracy and within-task ranking quality (AUC). Across SWE and mathematics, StarRM-9B matches or exceeds the domain-specific reward models in selection accuracy and achieves consistently strong within-task AUC. On SWE, it achieves the highest average accuracy and AUC, 72.3% and 0.588, compared with 69.8% and 0.529 for OpenHands-32B, despite its smaller size. The same advantage appears in mathematical reasoning: StarRM achieves 86.6% accuracy on MATH-500 and 23.3% on AIME 2025, with AUCs of 0.817 and 0.799, leading both the domain-specific reward models and LLM judges on these datasets.

Table 1: Reward-model evaluation on SWE-bench Verified. StarRM achieves the highest average BoN@16 accuracy (%) and within-task AUC across five generation models. Qwen denotes Qwen3.6, Luna/Terra denote GPT-5.6-Luna/Terra, and superscript J marks LLM judges.

Scorer	Qwen 27B		Qwen 35B		GPT-5.4		Luna		Terra		Average	
	BoN	AUC	BoN	AUC	BoN	AUC	BoN	AUC	BoN	AUC	BoN	AUC
Random	65.8	0.509	62.6	0.503	69.4	0.498	73.8	0.499	76.2	0.510	69.6	0.504
Majority vote	68.4	0.502	62.6	0.492	69.4	0.506	73.2	0.495	75.2	0.500	69.8	0.499
OpenHands-4B	64.0	0.457	59.2	0.472	68.6	0.511	71.2	0.438	75.4	0.493	67.7	0.474
DeepSWE-14B	66.6	0.581	64.8	0.564	68.8	0.488	70.6	0.485	75.4	0.504	69.2	0.524
OpenHands-32B	68.0	0.587	63.6	0.582	69.6	0.506	71.0	0.462	76.8	0.507	69.8	0.529
GPT-5.4^{J}	69.8	**0.615**	**68.0**	0.581	68.8	0.488	72.4	0.519	75.4	0.503	70.9	0.541
GPT-5.6-Luna^{J}	68.8	0.614	67.0	**0.628**	70.2	0.517	**75.6**	0.569	75.4	0.472	71.4	0.560
StarRM-9B	**70.2**	0.595	66.0	0.587	**72.0**	**0.582**	**75.6**	**0.597**	**77.8**	**0.579**	**72.3**	**0.588**

Table 2: Reward-model evaluation on mathematics. BoN@16 accuracy (%) and within-task AUC on ReTool trajectories.

Scorer	MATH-500		AIME 2024		AIME 2025	
	BoN	AUC	BoN	AUC	BoN	AUC
Random	74.8	0.498	13.3	0.487	10.0	0.506
Majority vote	85.2	0.499	26.7	0.508	16.7	0.519
Qwen Math-7B	80.0	0.647	23.3	0.507	13.3	0.500
ReasonFlux-7B	78.6	0.631	16.7	0.609	6.7	0.535
GenPRM-7B	83.0	0.722	30.0	0.665	16.7	0.542
GPT-5.4^{J}	83.4	0.769	30.0	0.814	20.0	0.739
GPT-5.6-Luna^{J}	82.4	0.795	**33.3**	**0.890**	20.0	0.795
StarRM-9B	**86.6**	**0.817**	30.0	0.798	**23.3**	**0.799**

Table 3: Generalization to unseen benchmarks. Accuracy without retraining

Task	Scorer	BoN	AUC
TB2	OpenHands-4B	38.2	0.368
	OpenHands-32B	48.3	0.651
	GPT-5.6-Luna^{J}	**58.4**	**0.686**
	StarRM-9B	57.3	0.670
LCB	Skywork-7B	79.4	0.462
	GPT-5.6-Luna^{J}	82.3	0.555
	StarRM-9B	**83.4**	**0.583**
CyberGym	OpenHands-4B	32.8	0.381
	OpenHands-32B	32.1	0.531
	GPT-5.6-Luna^{J}	40.3	0.587
	StarRM-9B	**50.0**	**0.840**

The gains also extend across generation models. On Qwen3.6-35B-A3B, it reaches 66.0%, exceeding the strongest domain-specific reward model's 64.8%. On trajectories from the stronger GPT-5.4, GPT-5.6-Luna, and GPT-5.6-Terra models, it attains 72.0%, 75.6%, and 77.8% accuracy, respectively, matching or exceeding both LLM judges while leading AUC in all three settings. The judges remain stronger in some settings: GPT-5.4 achieves 68.0% accuracy on Qwen3.6-35B-A3B trajectories, and GPT-5.6-Luna reaches 33.3% on AIME 2024 versus our 30.0%. Together, these results show that a compact 9B reward model can remain competitive with strong GPT-based judges while ranking and selecting trajectories across domains and generation models.

5 Analysis

5.1 Scoring Partial Trajectories

Figure 3 examines how ranking quality changes as more of a trajectory becomes visible. On SWE-bench Verified, StarRM's AUC increases from 0.495 at 25% progress to 0.578 at completion, exceeding DeepSWE-14B (0.548) and the GPT-5.4 judge (0.546) at the final checkpoint. On AIME 2025, AUC rises from 0.631 to a peak of 0.832 at 75% progress, and StarRM leads the compared evaluators at every checkpoint. The results indicate that reasoning and execution evidence helps the reward model distinguish promising trajectories, supporting its use for intermediate search decisions.

5.2 Generalization and Model Size

We evaluate StarRM on benchmarks not used for reward-model training, using the full Terminal-Bench 2 (Merrill et al., 2026) and LiveCodeBench (Jain et al., 2025a) evaluation sets and 134 selected CyberGym (Wang et al., 2026b) instances, without further training. StarRM achieves 57.3%, 83.4%, and 50.0% selection accuracy, respectively, exceeding the domain-specific reward baselines in Table 3.

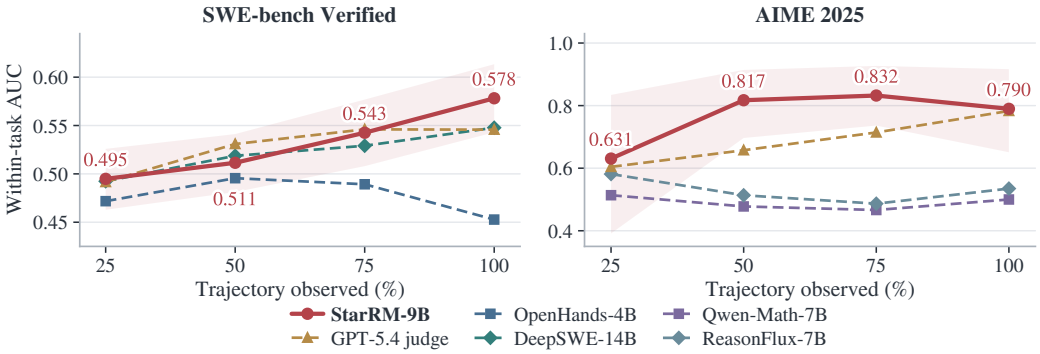


Figure 3: Ranking partial trajectories. StarRM's within-task AUC generally increases with more observed trajectory context. Shading denotes 95% task-bootstrap confidence intervals.

SWE-bench Verified			MATH-500		
Scorer	**BoN**	**Δ**	**Scorer**	**BoN**	**Δ**
OpenHands-4B	64.0	-4.0	Qwen Math-7B	80.0	-2.8
DeepSWE-14B	66.6	-1.4	ReasonFlux-7B	78.6	-4.2
OpenHands-32B	68.0	0.0	Granite Math-8B	82.8	0.0
StarRM-2B	69.2	+1.2	**StarRM-2B**	83.0	+0.2
StarRM-4B	69.4	+1.4	**StarRM-4B**	84.2	+1.4
StarRM-9B	**70.2**	+2.2	**StarRM-9B**	**86.6**	+3.8

Table 4: Effect of reward-model size. BoN@16 accuracy (%) increases from 2B to 9B; Δ measures percentage-point gains over the strongest listed baseline.

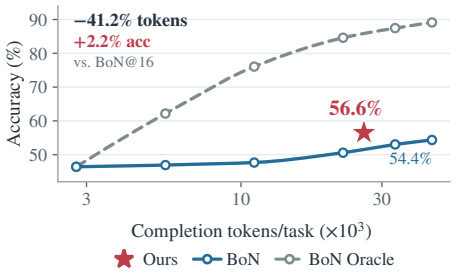


Figure 4: Test-time scaling with GPT-5.4. On SWE-bench Verified tasks, SPAR outperforms BoN@16 with 41.2% fewer tokens.

It also surpasses the GPT-5.6-Luna judge on LiveCodeBench and CyberGym, while remaining within 1.1 percentage points on Terminal-Bench 2. This transfer suggests that the learned comparisons extend to terminal interaction, code generation, and security tasks beyond the training domains.

We additionally train reward models from Qwen3.5-2B, 4B, and 9B (Table 4). Even the 2B model reaches 69.2% accuracy on SWE and 83.0% on MATH-500, exceeding the strongest listed external reward models by 1.2 and 0.2 percentage points. Increasing model size yields 69.4% and 84.2% at 4B, and 70.2% and 86.6% at 9B. Thus, useful ranking ability is retained at small model sizes, with the 9B variant providing the best selection accuracy.

5.3 Test-Time Scaling with GPT-5.4

We also evaluate SPAR using GPT-5.4 to generate agent trajectories. SPAR improves accuracy over BoN@16 by 2.2 percentage points while using 41.2% fewer generated completion tokens (Figure 4). These results show that SPAR also supports efficient test-time scaling with a stronger generation model.

6 Conclusion

We introduced SPAR, a reward-guided tree search framework for efficient agent test-time scaling. It reuses executed prefixes and reallocates computation to promising continuations through adaptive branching and pruning. To guide search and final selection, SPAR incorporates StarRM, a compact reward model trained with pairwise and listwise objectives. On SWE-bench Verified and AIME 2024, SPAR improves accuracy over BoN@16 with 46.3% and 58.4% fewer generated tokens, respectively. The framework also improves accuracy and efficiency with GPT-5.4 as the generation model, while StarRM-9B remains competitive with strong GPT-based judges and generalizes to unseen benchmarks. These results demonstrate how learned trajectory comparisons and prefix reuse can jointly improve the accuracy–cost tradeoff of long-horizon agents.

AI USE STATEMENT

Beyond the use of language models as experimental systems described in the paper, generative AI tools were used only to edit and polish the manuscript for grammar, clarity, and readability. They were not used to develop the research methodology, design or implement the experiments, analyze the data, or interpret the results. All AI-assisted edits were reviewed by the authors. The authors take full responsibility for the final content of this work.

REPRODUCIBILITY STATEMENT

We provide the details needed to reproduce our experiments throughout the paper and appendix. Section 4.1 specifies the benchmarks, models, baselines, and evaluation protocols. Appendix A.1 provides training and inference hyperparameters and agent configurations, Appendix A.2 describes the reward-model input formats, and Appendix A.3 provides the LLM-as-a-judge prompt. The benchmarks and open-weight models used in the main experiments are publicly available, and all models used in additional evaluations are explicitly identified.

REFERENCES

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Antonis Antoniades, Albert Örwall, Kexun Zhang, Yuxi Xie, Anirudh Goyal, and William Wang. Swe-search: Enhancing software agents with monte carlo tree search and iterative refinement. In *International Conference on Learning Representations*, volume 2025, pp. 64485–64515, 2025.

Ibragim Badertdinov, Alexander Golubev, Maksim Nekrashevich, Anton Shevtsov, Simon Karasik, Andrei Andriushchenko, Maria Trofimova, Daria Litvintseva, and Boris Yangel. Swe-rebench: An automated pipeline for task collection and decontaminated evaluation of software engineering agents. *Advances in Neural Information Processing Systems*, 38, 2026.

Edward Beeching, Lewis Tunstall, and Sasha Rush. Scaling test-time compute with open models. URL `https://huggingface.co/spaces/HuggingFaceH4/blogpost-scaling-test-time-compute`.

Hyungjoo Chae, Seonghwan Kim, Junhee Cho, Seungone Kim, Seungjun Moon, Gyeom Hwangbo, Dongha Lim, Minjin Kim, Yeonjun Hwang, Minju Gwak, et al. Web-shepherd: Advancing prms for reinforcing web agents. *Advances in Neural Information Processing Systems*, 38:63314–63356, 2026.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Yifeng Ding and Lingming Zhang. Swe-replay: Efficient test-time scaling for software engineering agents. *arXiv preprint arXiv:2601.22129*, 2026.

Ryan Ehrlich, Bradley Brown, Jordan Juravsky, Ronald Clark, Christopher Ré, and Azalia Mirhoseini. Codemonkeys: Scaling test-time compute for software engineering. *arXiv preprint arXiv:2501.14723*, 2025.

Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. Retool: Reinforcement learning for strategic tool use in llms. In *International Conference on Learning Representations*, volume 2026, pp. 37909–37926, 2026.

Alex Graves. Sequence transduction with recurrent neural networks. *arXiv preprint arXiv:1211.3711*, 2012.

Xinyu Guan, Li Lyna Zhang, Yifei Liu, Ning Shang, Youran Sun, Yi Zhu, Fan Yang, and Mao Yang. rstar-math: Small llms can master math reasoning with self-evolved deep thinking. *arXiv preprint arXiv:2501.04519*, 2025.

Hao Han, Jin Xie, Xuehao Ma, Weiquan Zhu, Ziyao Zhang, ZhiLiang Long, Hongkai Chen, and Qingwen Ye. Swe-trace: Optimizing long-horizon swe agents through rubric process reward models and heuristic test-time scaling. *arXiv preprint arXiv:2604.14820*, 2026.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Naman Jain, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, volume 2025, pp. 58791–58831, 2025a.

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

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

Jaewoo Lee, Archiki Prasad, Justin Chen, Zaid Khan, Elias Stengel-Eskin, and Mohit Bansal. Prints: Reward modeling for long-horizon information seeking. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 34120–34138, 2026a.

Nicholas Lee, Lutfi Eren Erdogan, Chris Joseph John, Surya Krishnapillai, Michael W Mahoney, Kurt Keutzer, and Amir Gholami. Agentic test-time scaling for webagents. *arXiv preprint arXiv:2602.12276*, 2026b.

Mukai Li, Qingcheng Zeng, Tianqing Fang, Zhenwen Liang, Linfeng Song, Qi Liu, Haitao Mi, and Dong Yu. Verified critical step optimization for llm agents. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 39627–39639, 2026a.

Xiaochuan Li, Ryan Ming, Pranav Setlur, Abhijay Paladugu, Andy Tang, Hao Kang, Shuai Shao, Rong Jin, and Chenyan Xiong. Benchmark test-time scaling of general llm agents. *arXiv preprint arXiv:2602.18998*, 2026b.

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

Zongyu Lin, Yao Tang, Xingcheng Yao, Da Yin, Ziniu Hu, Yizhou Sun, and Kai-Wei Chang. Qlass: Boosting language agent inference via q-guided stepwise search. *arXiv preprint arXiv:2502.02584*, 2025.

Liangchen Luo, Yinxiao Liu, Rosanne Liu, Samrat Phatale, Meiqi Guo, Harsh Lara, Yunxuan Li, Lei Shu, Yun Zhu, Lei Meng, et al. Improve mathematical reasoning in language models by automated process supervision. *arXiv preprint arXiv:2406.06592*, 2024.

Michael Luo, Naman Jain, Jaskirat Singh, Sijun Tan, Ameen Patel, Qingyang Wu, Alpay Ariyak, Colin Cai, Shang Zhu Tarun Venkat, Ben Athiwaratkun, et al. Deepswe: Training a state-of-the-art coding agent from scratch by scaling rl. *Notion Blog*, 2025a.

Michael Luo, Sijun Tan, Justin Wong, Xiaoxiang Shi, William Y Tang, Manan Roongta, Colin Cai, Jeffrey Luo, Tianjun Zhang, Li Erran Li, et al. Deepscaler: Surpassing o1-preview with a 1.5 b model by scaling rl. *Notion Blog*, 2025b.

MAA. American invitational mathematics examination (AIME). Mathematics Competition Series, n.d. URL `https://maa.org/math-competitions/aime`.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

Chenhui Mao, Yuanting Lei, Zhixiang Wei, Ming Liang, Zhixiang Wang, Jingxuan Xu, Dajun Chen, Wei Jiang, and Yong Li. Egss: Entropy-guided stepwise scaling for reliable software engineering. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 29481–29499, 2026.

Mike Merrill, Alexander Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Shin, Thomas Walshe, E Kelly Buchanan, et al. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *International Conference on Learning Representations*, volume 2026, pp. 40903–40986, 2026.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori B Hashimoto. s1: Simple test-time scaling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 20286–20332, 2025.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training software engineering agents and verifiers with swe-gym. *arXiv preprint arXiv:2412.21139*, 2024.

Qwen Team. Qwen3.6-27B: Flagship-level coding in a 27B dense model, April 2026a. URL https://qwen.ai/blog?id=qwen3.6-27b.

Qwen Team. Qwen3.6-35B-A3B: Agentic coding power, now open to all, April 2026b. URL https://qwen.ai/blog?id=qwen3.6-35b-a3b.

Junhong Shen, Hao Bai, Lunjun Zhang, Yifei Zhou, Amrith Setlur, Shengbang Tong, Diego Caples, Nan Jiang, Tong Zhang, Ameet Talwalkar, et al. Thinking vs. doing: Agents that reason by scaling test-time interaction. *arXiv preprint arXiv:2506.07976*, 2025.

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

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Maciej Świechowski, Konrad Godlewski, Bartosz Sawicki, and Jacek Mańdziuk. Monte carlo tree search: a review of recent modifications and applications: M. świechowski et al. *Artificial Intelligence Review*, 56(3):2497–2562, 2023.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. 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)*, pp. 9426–9439, 2024.

Xingyao Wang, Valerie Chen, Heng Ji, and Graham Neubig. A rubric-supervised critic from sparse real-world outcomes. *arXiv preprint arXiv:2603.03800*, 2026a.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

Zhun Wang, Tianneng Shi, Jingxuan He, Matthew Cai, Jialin Zhang, and Dawn Song. Cybergym: Evaluating ai agents' real-world cybersecurity capabilities at scale. In *International Conference on Learning Representations*, volume 2026, pp. 123341–123386, 2026b.

Yu Xia, Jingru Fan, Weize Chen, Siyu Yan, Xin Cong, Zhong Zhang, Yaxi Lu, Yankai Lin, Zhiyuan Liu, and Maosong Sun. Agentrm: Enhancing agent generalization with reward modeling. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 19277–19290, 2025.

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

Lifan Yuan, Wendi Li, Huayu Chen, Ganqu Cui, Ning Ding, Kaiyan Zhang, Bowen Zhou, Zhiyuan Liu, and Hao Peng. Free process rewards without process labels. *arXiv preprint arXiv:2412.01981*, 2024.

Guangtao Zeng, Maohao Shen, Delin Chen, Zhenting Qi, Subhro Das, Dan Gutfreund, David Cox, Gregory Wornell, Wei Lu, Zhang-Wei Hong, et al. Satori-swe: Evolutionary test-time scaling for sample-efficient software engineering. *arXiv preprint arXiv:2505.23604*, 2025.

Zhenru Zhang, Chujie Zheng, Yangzhen Wu, Beichen Zhang, Runji Lin, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. The lessons of developing process reward models in mathematical reasoning. *arXiv preprint arXiv:2501.07301*, 2025.

Jian Zhao, Runze Liu, Kaiyan Zhang, Zhimu Zhou, Junqi Gao, Dong Li, Jiafei Lyu, Zhouyi Qian, Biqing Qi, Xiu Li, et al. Genprm: Scaling test-time compute of process reward models via generative reasoning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 34932–34940, 2026.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning acting and planning in language models. *arXiv preprint arXiv:2310.04406*, 2023.

King Zhu, Hanhao Li, Siwei Wu, Tianshun Xing, Dehua Ma, Xiangru Tang, Minghao Liu, Jian Yang, Jiaheng Liu, Yuchen Eleanor Jiang, et al. Scaling test-time compute for llm agents. *arXiv preprint arXiv:2506.12928*, 2025.

Jiaru Zou, Ling Yang, Jingwen Gu, Jiahao Qiu, Ke Shen, Jingrui He, and Mengdi Wang. Reasonflux-prm: Trajectory-aware prms for long chain-of-thought reasoning in llms. *Advances in Neural Information Processing Systems*, 38:26764–26802, 2026.

A Appendix

A.1 Additional Implementation Details

We train LoRA (Hu et al., 2021) adapters with rank 32 and $\alpha = 64$, using an 8,192-token input limit, a learning rate of 5×10^{-5}, a batch size of 16, and a listwise loss weight of $\lambda = 0.5$. Training examples from completed trajectories, candidate actions, and prefixes are sampled in a $5{:}4{:}1$ ratio. For trajectory generation, we use temperature 0.8 and top-$p = 0.95$. Mathematical agents use a Python interpreter with limits of 20 turns and 3,072 output tokens per turn, while SWE agents use mini-SWE-agent in Docker with limits of 80 turns and 8,192 output tokens per turn.

A.2 Reward-Model Input Examples

We train StarRM on three types of input: completed trajectories, prefixes, and candidate actions with their observations. The examples below illustrate each format, including the task, relevant interaction history, and task-success question; ... denotes omitted earlier turns.

Input 1: Completed trajectory

Task

Find all real solutions of $x^2 - 5x + 6 = 0$.

Interaction history

... *(earlier turns omitted)*

Agent: I will check whether 2 and 3 have the sum and product needed to factor the polynomial.

Python: `print(2 + 3, 2 * 3)`

Observation: `5 6`

Agent: This gives $(x-2)(x-3)$. I will substitute both roots into the original expression.

Python: `print([x*x - 5*x + 6 for x in (2, 3)])`

Observation: `[0, 0]`

Agent: Both values satisfy the equation. Since $(x-2)(x-3)=0$, these are the only solutions.

Final answer

$x = 2$ or $x = 3$.

Question

Did this attempt actually solve the task? Answer + for yes or − for no.

Verdict:

Input 2: Trajectory prefix

Task

Find all real solutions of $x^2 - 5x + 6 = 0$.

Interaction history so far

. . . (earlier turns omitted)

Agent: I will check whether 2 and 3 have the sum and product needed to factor the polynomial.

Python: `print(2 + 3, 2 * 3)`

Observation: `5 6`

Remaining budget

12 agent turns.

Question

Will this attempt, continuing from here, end up solving the task? Answer + for yes or − for no.

Verdict:

Input 3: Candidate action with its observation

Task

Find all real solutions of $x^2 - 5x + 6 = 0$.

Parent history

. . . (earlier turns omitted)

Agent: I will check whether 2 and 3 have the sum and product needed to factor the polynomial.

Python: `print(2 + 3, 2 * 3)`

Observation: `5 6`

Candidate action and resulting observation

Agent: This gives $(x-2)(x-3)$. I will substitute both roots into the original expression.

Python: `print([x*x - 5*x + 6 for x in (2, 3)])`

Observation: `[0, 0]`

Remaining budget

11 agent turns.

Question

The agent took the candidate action above and saw its result. Continuing from there, will it solve the task? Answer + for yes or − for no.

Verdict:

A.3 LLM-AS-A-JUDGE PROMPT

The template below summarizes the rubric used by our LLM judges to evaluate completed trajectories. The evaluator converts the judge's assessments of task completion and supporting evidence into a scalar trajectory score.

LLM judge: Completed-trajectory evaluation

Instruction

Evaluate whether the agent completed the task. Base your assessment on the supplied reasoning, actions, observations, and final result. Do not assume access to a reference answer or hidden test results.

Task

[Task description]

Task requirements

[Required outcomes and constraints extracted from the task, each with an ID]

Agent trajectory

[Complete sequence of agent reasoning, actions, and observations, with step indices]

Final result

[Final answer and, for repository tasks, the submitted patch]

Evaluation rubric

1. **Check each requirement.** Mark it as *satisfied* if fully established, *partial* if only partly established, *contradicted* if the evidence conflicts with it, or *unknown* if the evidence is insufficient.
2. **Assess the supporting evidence.** Distinguish no evidence, an unsupported claim, a direct check, an independent check, and coverage of relevant boundary or regression cases. For mathematics, inspect derivations, substitutions, and domain constraints. For repository tasks, inspect the final patch and test outputs after the last edit.
3. **Avoid unsupported conclusions.** A statement that a test passed is not an observed test result. One successful example does not establish all required behavior. Cite an exact quotation and its source for every status other than unknown; do not infer failures from speculation.
4. **Record direct evidence of problems.** Report unresolved failures, failures after the final edit, conflicts with the task's scope, fabricated evidence, or unsafe actions only when a supplied artifact directly establishes them.

Return

For each requirement, give its ID, status, evidence strength, supporting quotations and sources where available, and a brief explanation. Identify any directly supported problems and finish with a two-sentence summary.