

ASE-MCTS: Asynchronous Self-Evaluated Monte Carlo Tree Search for Efficient LLM Reasoning

Yang Wu^{1,2,*}, Yuyang Liu^{1,2,*}, Zhicheng Zhang^{2,1,*}, Xiaolong Zheng^{1,3,†}

¹State Key Laboratory of Multimodal Artificial Intelligence Systems,
Institute of Automation, Chinese Academy of Sciences

²School of Artificial Intelligence, University of Chinese Academy of Sciences

³University of Chinese Academy of Sciences

Beijing, China

{wuyang2022, liuyuyang2025, zhangzhicheng2024, xiaolong.zheng}@ia.ac.cn

Abstract

Monte Carlo Tree Search (MCTS) is a promising test-time method for improving LLMs on hard reasoning problems, where correct trajectories are often sparse. Self-evaluated MCTS is attractive because it avoids external reward models, verifiers, and stronger judges, but it faces a central paradox: MCTS depends on reliable intermediate rewards, while LLM self-evaluation is noisy and unstable. We propose ASE-MCTS, an **A**synchronous **S**elf-Evaluated **M**C**T**S framework. ASE-MCTS uses search-oriented reward analysis to derive an effective composition of rollout, process, and pairwise rewards from their global and local search roles. It further combines flexible search-space control with dependency-aware asynchronous execution, reducing redundant exploration and latency. Experiments on mathematical and scientific reasoning benchmarks show that ASE-MCTS improves reasoning without external supervision, especially on hard sparse-trajectory problems, and achieves up to a $6\times$ wall-clock speedup over Sync-MCTS in the branch-factor scalability experiment.

1 Introduction

Large Language Models (LLMs) have demonstrated strong reasoning abilities, yet they still struggle with complex mathematical and scientific problems where correct reasoning trajectories are sparse. Large Reasoning Models such as OpenAI-o1 (OpenAI, 2024) and Chain-of-Thought prompting (Wei et al., 2022) improve single-trajectory reasoning, while inference-time methods such as Self-Consistency (Wang et al., 2023) and Best-of-N (Cobbe et al., 2021) broaden coverage through repeated sampling. However, these sampling-based strategies depend on correct answers appearing frequently enough to dominate a vote or a selector. When plausible but incorrect trajectories occupy most of the sampling distribution, simple aggregation quickly saturates.

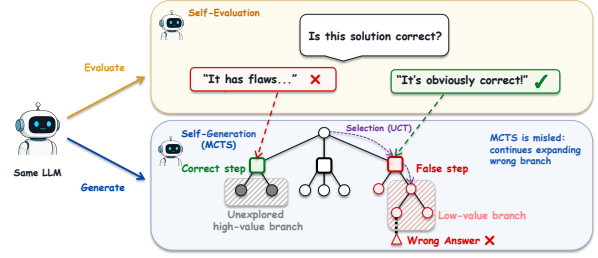


Figure 1: The paradox of self-evaluated MCTS: Noisy reward signal leads to wrong branches.

Monte Carlo Tree Search (MCTS) is a natural alternative for this sparse-correct-trajectory regime. By searching over partial reasoning states, MCTS can direct computation toward promising branches rather than repeatedly sampling complete solutions (Hao et al., 2023). Yet this advantage relies on reliable intermediate value estimates (Guo et al., 2025). Existing systems often obtain them from external reward models, verifiers, or stronger judges (Chen et al., 2025; Lightman et al., 2024; Tian et al., 2024), introducing extra supervision, deployment cost, VRAM usage, and latency.

We focus on the constrained setting of fully self-evaluated MCTS, where the same policy model generates and evaluates its own reasoning paths. Although this setting has been explored in prior MCTS-style reasoning systems (Hao et al., 2023; Wu et al., 2025), it exposes a central unresolved paradox: MCTS requires reliable intermediate rewards to guide search, but LLM self-evaluation is often noisy, biased, and poorly calibrated (Huang et al., 2024; Zheng et al., 2023; Chen et al., 2024). This issue is especially severe for small and medium-sized models, where unstable self-assessment can steer search toward plausible but incorrect branches.

Beyond this, an effective self-evaluated MCTS system also requires careful search and execution design. On the search side, reasoning steps and ac-

tion spaces must be flexible enough to adapt to different tasks, while remaining controlled enough to avoid redundant or invalid exploration. On the execution side, self-evaluated MCTS requires frequent LLM calls for generation and evaluation (Snell et al., 2025), and executing these heterogeneous calls synchronously creates severe blocking and limits practical usability.

To address these challenges, we propose ASE-MCTS, an **Asynchronous Self-Evaluated MCTS** framework for LLM reasoning. ASE-MCTS controls tree growth through a flexible prompt-guided search space and semantic terminal detection, preserving expressive reasoning steps while reducing redundant expansion. To mitigate the self-evaluation paradox, we propose **Search-Oriented Reward Analysis**, which analyzes intrinsic reward signals through global consistency and local discriminability to derive a more reliable reward composition. Finally, a dependency-aware asynchronous orchestrator parallelizes independent generation and evaluation calls, substantially reducing wall-clock latency. Experiments on mathematical and scientific reasoning benchmarks show that ASE-MCTS improves hard reasoning performance, especially when correct trajectories are sparse, while achieving favorable token-cost and latency trade-offs over existing search baselines.

Our contributions are summarized as follows:

- We propose **ASE-MCTS**, a fully self-evaluated MCTS framework that uses flexible search-space control and semantic terminal detection to support expressive and efficient tree search.
- We propose **Search-Oriented Reward Analysis** to address the self-evaluation paradox, deriving a practical composition of rollout, process, and pairwise signals from the perspectives of global consistency and local discriminability.
- We re-architect the MCTS workflow as a DAG managed by an **Asynchronous Orchestrator**. This design maximizes parallelism among generation and evaluation tasks in self-evaluated MCTS, effectively improving inference efficiency.

2 Related Work

LLM Reasoning and Search Enhancing LLM reasoning capabilities primarily hinges on decom-

posing complex problems into step-by-step rationales (Wei et al., 2022). To broaden the exploration of the solution space, ensemble strategies like Self-Consistency (Wang et al., 2023) and Best-of-N (Cobbe et al., 2021) employ parallel sampling and post-hoc aggregation, while iterative methods such as Self-Refine (Madaan et al., 2023) introduce mechanisms for intrinsic error correction. Furthermore, tree-search algorithms like Tree of Thoughts (ToT) (Yao et al., 2023; Misaki et al., 2025) transform reasoning into a structured search task, dynamically allocating resources to the most promising reasoning paths.

MCTS for LLMs Due to its superior exploration-exploitation balance, MCTS has been widely adapted for LLM reasoning. Unlike approaches that leverage MCTS primarily for synthesizing high-quality training data (Tian et al., 2024; Guan et al., 2025), test-time strategies such as RAP (Hao et al., 2023) and AlphaLLM (Tian et al., 2024) guide inference via intrinsic reward signals derived from self-evaluation or consistency. To enhance reward reliability, recent works like rStar (Qi et al., 2025) further incorporate diverse action spaces and mutual consistency verification to identify and select high-quality reasoning trajectories.

Efficient Tree Search Although tree search significantly enhances reasoning capabilities, it entails prohibitive token consumption and time costs. To mitigate these overheads, DEFT (Yao et al., 2024) optimizes memory utilization, while FastMCTS (Li et al., 2025) reduces ineffective exploration through an Adaptive Stay Policy. Complementing these algorithmic improvements, frameworks like DPTS (Ding et al., 2025) and WU-UCT (Liu et al., 2020) focus on accelerating inference via advanced parallelization strategies.

3 ASE-MCTS Framework

This section describes the design of ASE-MCTS around three requirements of self-evaluated tree search. First, the search space must be flexible enough to express task-dependent reasoning steps, while remaining constrained enough to avoid redundant or uncontrolled expansion. Second, the reward signal must turn noisy self-evaluation into useful search guidance. Third, the implementation must execute the many heterogeneous LLM calls efficiently. Accordingly, ASE-MCTS consists of search space control, search-oriented reward analy-

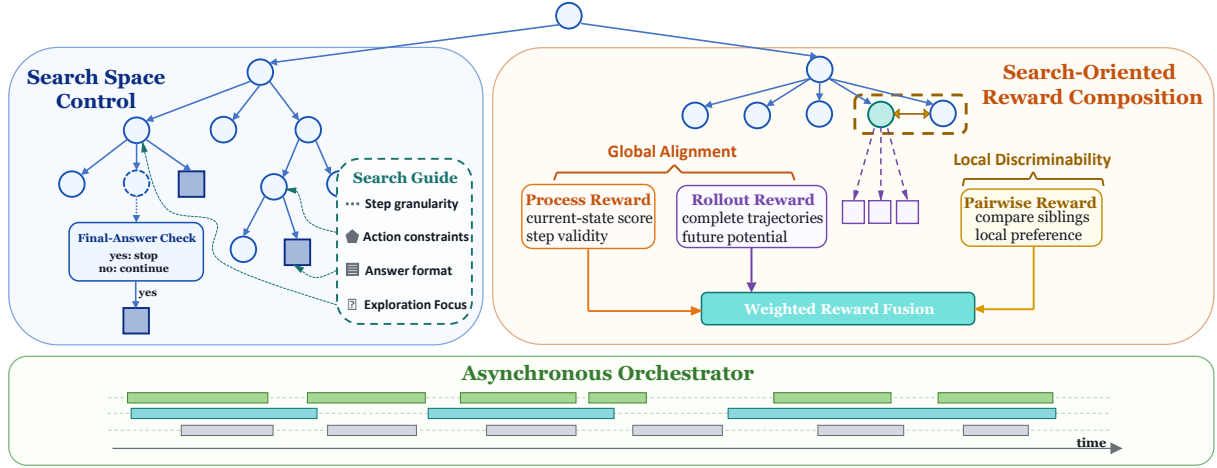


Figure 2: Overview of ASE-MCTS. Search space control guides flexible but constrained tree expansion; search-oriented reward composition combines process, rollout, and pairwise signals for robust self-evaluation; and the asynchronous orchestrator overlaps independent generation and evaluation tasks to improve efficiency.

sis, and a dependency-aware asynchronous orchestrator. We otherwise follow the standard MCTS backbone of selection, expansion, evaluation, and backup, as detailed in Appendix B.1; this section focuses on the core designs specific to ASE-MCTS.

3.1 Search Space Control

LLM-based MCTS requires reasoning actions that are expressive enough for task-dependent steps, yet constrained enough to avoid uncontrolled growth and redundant exploration. ASE-MCTS controls the search space through two designs: a prompt-guided action space for expansion and a semantic terminal-control mechanism for early stopping.

Flexible Prompt-Guided Action Space Rather than hard-coding fixed action types, ASE-MCTS treats each expansion as a natural-language reasoning step generated under task-specific guidance. Fixed action types can make the search easier to control, but they may also impose an artificial reasoning granularity: a coherent step can be split into several mechanical actions, increasing tree depth, token cost, and evaluation overhead. In ASE-MCTS, a lightweight “Search Guide” specifies step granularity, action constraints, exploration focus, and answer format in natural language. It also encourages candidate children to follow distinct reasoning directions under the same context, reducing near-duplicate sibling nodes without introducing a separate expansion module. This preserves the expressive capacity of each node while allowing the same MCTS backbone to adapt across mathematical and scientific reasoning tasks.

Semantic Terminal Control To avoid redundant expansion, ASE-MCTS separates step generation from terminal-state detection. After each expansion, the model produces a short final-answer check indicating whether the current state should be treated as terminal. This check is used only for stopping control and is not appended to the reasoning trajectory. In implementation, this terminal signal is produced inline with the expansion output, so it does not require an additional LLM call. By detecting terminal states early, ASE-MCTS avoids continuing from already complete states and improves search efficiency.

3.2 Search-Oriented Reward Analysis

The central challenge of self-evaluated MCTS is that search depends on intermediate rewards, while LLM self-evaluation is noisy and unstable. ASE-MCTS therefore does not rely on a single self-evaluation score. Instead, we analyze candidate reward signals according to their roles in MCTS and compose complementary signals into a more robust value estimate.

The Dual Role of Reward In MCTS, a reward signal is not merely a final correctness score. It must support two search decisions at intermediate states. First, it should provide *global alignment*, meaning that higher-valued states should be more likely to lead to correct final solutions. Second, it should provide *local discriminability*, meaning that the reward should distinguish among sibling nodes generated under the same context. These two roles are related but not equivalent: a signal

may be globally correlated with solution quality but weak at ranking similar sibling steps, or locally useful but poorly calibrated across different parts of the tree.

ET-based Reward Analysis Existing MCTS-based LLM reasoning methods have explored various reward signals, but it remains unclear which signal is useful for which part of the search process. Since intermediate reasoning steps do not have direct ground-truth labels, we construct Ensemble Truth (ET) as a consensus-reference value for analysis. ET aggregates normalized judgments from several strong LLMs to estimate the relative quality of sampled intermediate steps, while human ratings are collected independently as an external reference for analysis. Intuitively, a better reward signal should be more consistent with ET.

We use ET to diagnose candidate intrinsic signals through two metrics: Pearson correlation with ET for global alignment, and sibling-level regret for local discriminability. For each sibling set, let s_{ET}^* be the sibling preferred by ET and s_ϕ^* be the sibling selected by a candidate signal ϕ . The regret is defined as

$$R(\phi) = V_{\text{ET}}(s_{\text{ET}}^*) - V_{\text{ET}}(s_\phi^*).$$

It measures the value loss caused by a signal’s ranking error. Section 4.3 reports this analysis and shows that different reward signals provide complementary strengths.

Reward Signal Composition Guided by the above analysis, ASE-MCTS uses three complementary intrinsic signals: rollout reward, process reward, and pairwise reward. They evaluate an intermediate state from future potential, immediate step validity, and local sibling preference, respectively.

Rollout reward evaluates an intermediate state by the quality of complete trajectories rolled out from it. Starting from state s , the model samples N continuations $\{y_1, \dots, y_N\}$ and uses the average score of the completed solutions as the value estimate of s :

$$V_{\text{roll}}(s) = \frac{1}{N} \sum_{k=1}^N \text{Score}(s \rightarrow y_k).$$

It measures the future potential of the current node rather than only the local validity of the current step.

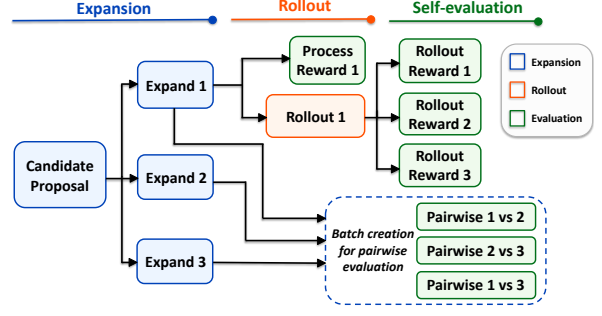


Figure 3: Dependency-aware task graph of ASE-MCTS. Expansion, rollout, and self-evaluation tasks become ready according to data dependencies, and ready tasks can be batched or executed concurrently instead of following a blocking stage-by-stage loop.

Process reward directly evaluates the quality of the current intermediate node without rollout:

$$V_{\text{proc}}(s) = \text{Score}(s).$$

It focuses on whether the reasoning state represented by s contains errors, violates task constraints, or deviates from a plausible path toward the final solution.

Pairwise reward evaluates the local preference of an intermediate state among sibling candidates under the same parent context. For each state s_i , we compare it against the other sibling states and aggregate the resulting preferences using a Borda-style score. It measures local discriminability by indicating which sibling is more promising under the same context (Zhang et al., 2025). The detailed computation is provided in Appendix B.2.

Before fusion, all reward signals are normalized to $[0, 1]$ to ensure a common numerical scale. The final self-evaluated value is a weighted composition:

$$Q(s) = \alpha V_{\text{roll}}(s) + \beta V_{\text{proc}}(s) + \gamma V_{\text{pair}}(s).$$

Following the reward analysis, we use larger weights for rollout and process rewards, which provide stronger global alignment, and a smaller weight for pairwise reward, which mainly improves local sibling discrimination.

3.3 Asynchronous Orchestrator

Self-evaluated MCTS introduces many heterogeneous LLM calls, including expansion, rollout, process evaluation, rollout evaluation, and pairwise comparison. A standard synchronous implementation executes these calls in a fixed order, so short

tasks often wait for long rollouts or unrelated evaluations, leaving the inference engine underutilized. However, many tasks created during one MCTS iteration are not causally dependent on each other. For example, rollouts from different child nodes and reward evaluations for different trajectories can be executed independently.

ASE-MCTS therefore represents the search procedure as a dependency-aware task graph. Each node in the graph is an atomic LLM task, and each edge denotes a required data dependency. A task is scheduled as soon as all its predecessors are finished, rather than waiting for the entire procedural stage to complete. In this way, the orchestrator preserves the logical semantics of MCTS while allowing independent expansion, rollout, and evaluation tasks to run concurrently.

Figure 3 shows the dependency structure exploited by the orchestrator. Instead of executing MCTS as a blocking sequence of stages, ASE-MCTS maintains a frontier of ready LLM tasks and dispatches each task as soon as its inputs are available. This allows the heterogeneous generation and evaluation calls in self-evaluated MCTS to overlap across branches, reducing wall-clock latency without changing the underlying search semantics.

4 Experiments

Models. We evaluate ASE-MCTS on three open-source SLMs with increasing reasoning capability: Llama3.1-8B-Instruct (Grattafiori et al., 2024), Qwen2.5-7B-Instruct (Yang et al., 2024), and Qwen3-8B (Yang et al., 2025). Additional results on a stronger model DeepSeek-V3 are reported in Appendix C.3.

Datasets. We evaluate mathematical reasoning on MATH500 (Hendrycks et al., 2021), AMC2023 (AI-MO, 2024b), and AIME2024 (AI-MO, 2024a), which range from standard math benchmarks to high-difficulty competition problems. For scientific reasoning, we use GPQA-Diamond (Rein et al., 2024) and Olympiad-Physics (He et al., 2024), covering graduate-level multidisciplinary questions and complex physics derivation tasks. Together, these benchmarks allow us to test whether ASE-MCTS is effective when correct reasoning trajectories are sparse.

Baselines. We compare ASE-MCTS with both sampling-based and tree-search baselines. For

sampling-based methods, we include Zero-shot CoT (Wei et al., 2022), Self-Consistency (SC) (Wang et al., 2023), and Best-of-N (Cobbe et al., 2021). SC aggregates sampled answers by majority voting, while Best-of-N selects among sampled trajectories using self-evaluation. For tree search, we implement Tree of Thoughts (ToT) (Yao et al., 2023) as a BFS-based beam search guided by self-evaluation. We also compare with rStar (Qi et al., 2025), a representative MCTS-style reasoning method, including both its generator-majority and discriminator-guided variants under the original hyper-parameter setting.

4.1 Overall Performance on Hard Reasoning

Overall results. Table 1 shows that ASE-MCTS achieves the strongest or competitive performance across different backbones, with the clearest gains on high-difficulty benchmarks such as GPQA-Diamond and Olympiad-Physics. These results support our motivation: when correct trajectories are sparse, repeated sampling can quickly saturate, while ASE-MCTS can guide computation toward promising partial reasoning states and improve beyond conventional sampling baselines.

Baseline limitations. Self-Consistency remains a strong baseline, but it relies on correct answers appearing frequently enough to dominate the vote; we analyze this effect in Section 4.2. ToT is sensitive to noisy intermediate self-evaluation, since beam search may prematurely keep or discard branches based on unreliable local scores. For rStar, we observe sensitivity to generator-side repeated sampling and generator-discriminator capability matching; see Appendix C.2 for details. By composing rollout, process, and pairwise signals for different search roles, ASE-MCTS provides more robust guidance without external stronger judges.

4.2 Effectiveness under Sparse Correct Trajectories

The main results suggest that ASE-MCTS is most beneficial on difficult benchmarks, but this aggregate view does not explain *when* search helps. Our motivation is that repeated sampling becomes unreliable when correct trajectories are sparse: even if the model can occasionally produce a valid solution, the correct answer may not appear frequently enough to dominate majority voting.

To test this hypothesis, we compare ASE-MCTS with Self-Consistency across difficulty groups

Table 1: Comparison of reasoning accuracy across different models and tasks. **Bold** indicates the best result, and underline indicates the second best result.

Method	MATH500	AMC2023	AIME2024	GPQA-Diamond	Olympiad-physics
<i>Llama3.1-8B-Instruct</i>					
Zero-shot CoT	46.40	25.00	3.33	22.73	4.66
Best-of-N@8	48.00	17.50	3.33	24.24	7.63
Best-of-N@16	49.80	27.50	3.33	29.29	6.36
Self-Consistency@maj8	60.60	30.00	3.33	32.32	9.32
Self-Consistency@maj16	<u>62.60</u>	<u>35.00</u>	13.33	33.33	10.17
Self-Consistency@maj32	64.80	32.50	<u>10.00</u>	<u>32.83</u>	<u>11.86</u>
ToT(beam search)	35.60	20.00	0.00	16.16	5.08
rStar(generator @maj)	55.60	12.50	0.00	28.79	6.36
rStar(Qwen2.5-7B-Instruct)	51.60	22.50	0.00	30.81	0.85
ASE-MCTS	59.40	37.50	13.33	33.33	13.14
<i>Qwen2.5-7B-Instruct</i>					
Zero-shot CoT	73.20	50.00	10.00	34.85	12.29
Best-of-N@8	75.20	57.50	10.00	36.87	13.98
Best-of-N@16	74.60	52.50	13.33	38.38	13.98
Self-Consistency@maj8	78.20	60.00	<u>16.67</u>	35.35	17.80
Self-Consistency@maj16	80.20	<u>62.50</u>	13.33	37.37	18.64
Self-Consistency@maj32	<u>81.80</u>	<u>62.50</u>	<u>16.67</u>	35.86	<u>19.07</u>
ToT(beam search)	62.40	37.50	6.67	32.83	11.72
rStar(generator @maj)	70.80	45.00	13.33	35.35	11.02
rStar(Llama3.1-8B-Instruct)	64.80	42.50	6.67	30.30	3.81
ASE-MCTS	82.60	65.00	20.00	40.91	20.76
<i>Qwen3-8B</i>					
Zero-shot CoT	82.60	65.00	26.67	48.99	19.07
Best-of-N@8	87.40	75.00	36.67	55.56	23.31
Best-of-N@16	86.60	70.00	40.00	55.05	22.46
Self-Consistency@maj8	89.60	<u>77.50</u>	36.67	53.54	27.97
Self-Consistency@maj16	90.20	80.00	<u>40.00</u>	55.55	30.51
Self-Consistency@maj32	91.20	80.00	<u>40.00</u>	<u>56.57</u>	30.93
ToT(beam search)	65.80	42.50	13.33	33.84	13.98
rStar(generator @maj)	89.00	50.00	20.00	46.97	15.25
rStar(Qwen2.5-7B-Instruct)	78.20	27.50	10.00	36.36	6.78
ASE-MCTS	<u>90.60</u>	80.00	43.33	58.08	34.75

on GPQA-Diamond and Olympiad-Physics. As shown in Figure 4, ASE-MCTS is comparable to SC on easier problems, where correct trajectories are already frequent enough for voting to work. The gap becomes larger in the medium-to-hard groups, where SC rapidly deteriorates as incorrect trajectories dominate the sample pool. In contrast, ASE-MCTS remains more stable by searching over partial states and using intermediate self-assessment to allocate computation toward promising branches. These results show that ASE-MCTS is not merely a stronger sampler; it is most useful in the sparse-correct-trajectory regime where independent sampling begins to saturate.

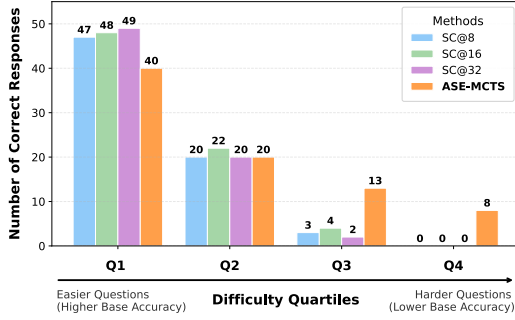
4.3 Search Guidance from Reward Signals

We further examine whether noisy self-assessment signals can provide reliable guidance for MCTS. As described in Section 3.2, ET is used only as an offline consensus reference for diagnosing intrinsic

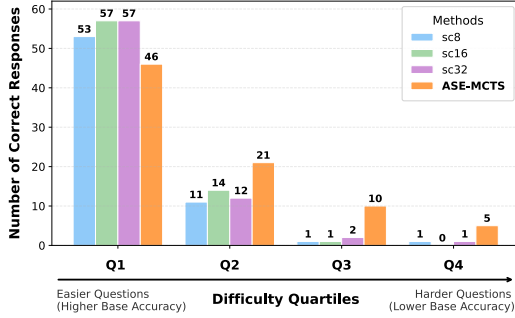
reward signals, not as an inference-time reward.

ET-based signal analysis. Figure 5 shows the agreement among value assessments from different LLMs and human experts. The fused ET score achieves the highest correlation with individual references, suggesting that ET provides a more stable consensus view than any single judge. This makes it suitable for diagnosing whether a reward signal captures meaningful quality differences among intermediate reasoning states.

Table 2 reports the global-local diagnostic results. Rollout and process rewards generally show stronger Pearson correlation with ET, indicating better global consistency with intermediate-state quality. In contrast, pairwise reward has weak global correlation but consistently achieves low regret, especially on GPQA-Diamond, showing its advantage in local sibling discrimination. Fusion improves global consistency while retaining com-



(a) Qwen2.5-7B-Instruct on GPQA-Diamond



(b) Qwen3-8B on Olympiad-physics

Figure 4: **Performance analysis across difficulty quartiles.** (a) Results using *Qwen2.5-7B-Instruct* on the *GPQA-Diamond* dataset. (b) Results using *Qwen3-8B* on the *Olympiad-physics* dataset. Questions are categorized into four quartiles (Q1–Q4) based on base accuracy. **ASE-MCTS** mostly outperforms baselines, particularly on the harder questions (Q3 and Q4).

petitive local regret, supporting our design of combining globally oriented rewards with a lightweight local comparison signal.

End-to-end reward ablation. We further evaluate whether the above diagnostic pattern translates into end-to-end search performance. Following the sparse-trajectory analysis, we conduct this controlled ablation on a 158-case medium-to-high-difficulty subset of GPQA-Diamond, where correct trajectories are rare and effective reward guidance is most critical. As shown in Table 3, individual signals provide usable but limited guidance, and pairwise reward alone performs worst, consistent with its role as a locally oriented rather than globally calibrated signal. Combining rollout and process rewards reaches 36.08% accuracy, while the best three-signal setting reaches 37.34%, a modest 1.26-point improvement. These results are consistent with pairwise comparison providing incremental local discrimination, but they also indicate that the end-to-end benefit of reward fusion is sensitive to the composition weights rather than uniformly

Signal	AMC2023		GPQA-Diamond	
	Pearson(↑)	Regret(↓)	Pearson(↑)	Regret(↓)
<i>Qwen2.5-7B-Instruct</i>				
Process	0.46	0.2201	0.36	0.2584
Roll-reward	0.53	0.3086	0.33	0.3180
Pairwise	0.07	0.2185	0.08	0.2673
Fusion	0.58	0.2216	0.40	0.2846
<i>Qwen3-8B</i>				
Process	0.51	0.2973	0.55	0.2382
Roll-reward	0.60	0.2949	0.44	0.3535
Pairwise	0.22	0.2591	0.14	0.2250
Fusion	0.65	0.2719	0.57	0.2287

Table 2: Global-local analysis of intrinsic reward signals on AMC2023 and GPQA-Diamond. Pearson measures global consistency with ET, while regret measures local sibling-selection error.

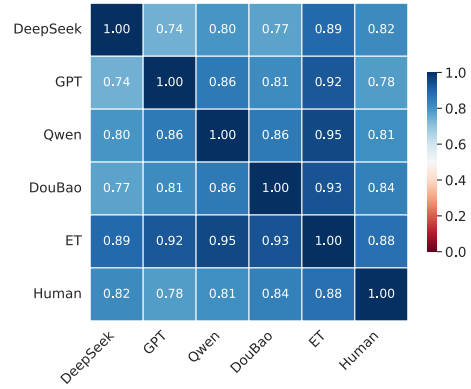


Figure 5: Pearson correlation heatmap of value assessments from different LLMs and human experts. The fused ET score provides a stable consensus reference for search-oriented reward analysis.

large.

4.4 Latency Efficiency and Token-Cost Trade-off

Finally, we evaluate the efficiency of ASE-MCTS from two perspectives: whether the asynchronous orchestrator reduces wall-clock latency, and whether the performance gains come from brute-force token consumption.

Latency reduction. We evaluate runtime on a 48GB NVIDIA RTX 5880 Ada GPU with fixed random seeds, using vLLM as the backend (Kwon et al., 2023). We compare ASE-MCTS with ToT, rStar(generator), and Sync-MCTS, a synchronous variant of our method. For a controlled comparison, we cap the number of LLM calls at approximately 180 for each method. As shown in Table 4, ASE-MCTS consistently achieves the lowest wall-clock latency across models and datasets. Compared with

Value Estimate $Q(s)$	Acc.
V_{roll}	34.18
V_{proc}	35.44
V_{pair}	31.65
$0.5V_{\text{roll}} + 0.5V_{\text{proc}}$	36.08
$0.8V_{\text{roll}} + 0.2V_{\text{pair}}$	35.44
$0.8V_{\text{proc}} + 0.2V_{\text{pair}}$	35.44
$0.4V_{\text{roll}} + 0.4V_{\text{proc}} + 0.2V_{\text{pair}}$	36.71
$0.4V_{\text{roll}} + 0.5V_{\text{proc}} + 0.1V_{\text{pair}}$	37.34

Table 3: End-to-end ablation of self-assessment value composition on a 158-case medium-to-high-difficulty subset of GPQA-Diamond.

Model	Method	AMC2023	GPQA-Diamond
		Time (s)	Time (s)
Llama-3.1 8B	ToT	526.24	606.56
	rStar(generator)	1219.84	1413.13
	Sync-MCTS	825.17	1132.32
	ASE-MCTS	295.64	281.13
Qwen-2.5 7B	ToT	460.53	402.49
	rStar(generator)	1593.76	1569.24
	Sync-MCTS	909.32	616.63
	ASE-MCTS	187.69	167.64
Qwen-3 8B	ToT	893.88	1754.83
	rStar(generator)	1845.67	1695.42
	Sync-MCTS	852.20	643.88
	ASE-MCTS	193.13	225.16

Table 4: Average computation time in seconds under a comparable LLM-call budget. ASE-MCTS consistently reduces wall-clock latency across models and datasets.

Sync-MCTS, ASE-MCTS achieves a $2.8\times$ – $4.8\times$ wall-clock speedup across models and datasets, showing that asynchronous execution can substantially reduce the wall-clock overhead of generation and self-evaluation tasks. ASE-MCTS is also substantially faster than ToT and rStar(generator), despite using richer reward signals for search guidance.

We further test scalability as the tree expands with larger branch factors. Figure 6 shows that Sync-MCTS becomes much slower as the number of LLM calls increases due to additional generation and evaluation tasks. In contrast, ASE-MCTS keeps latency growth relatively flat by parallelizing ready tasks. At branch factor 8, ASE-MCTS achieves a $6.1\times$ speedup over Sync-MCTS.

Accuracy-token trade-off. We also compare accuracy and average token cost on GPQA-Diamond. As shown in Table 5, SC@32 is token-efficient, but its accuracy is limited on difficult problems where correct trajectories are sparse. ASE-MCTS

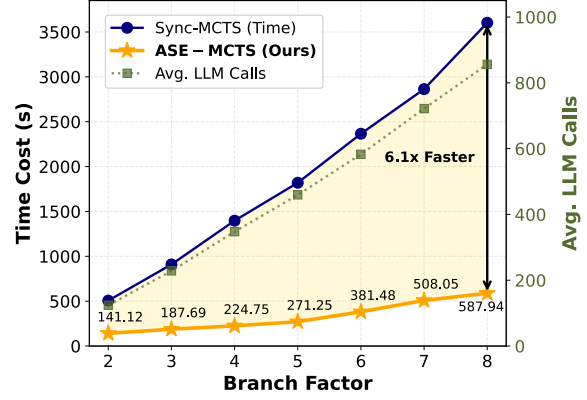


Figure 6: Latency comparison between Sync-MCTS and ASE-MCTS under varying branch factors on AMC2023.

Method	Acc.	Tokens	Token Ratio
SC@32	35.86	24.57k	$0.23\times$
ToT	32.83	451.23k	$4.19\times$
rStar@maj	35.35	1295.79k	$12.02\times$
ASE-MCTS	40.91	107.81k	$1.00\times$

Table 5: Accuracy-token trade-off on GPQA-Diamond. Token ratio is normalized by ASE-MCTS.

uses more tokens than SC, but this additional cost brings a clear accuracy gain. Compared with tree-search baselines, ASE-MCTS achieves higher accuracy with much lower token cost, using about $4.2\times$ fewer tokens than ToT and $12.0\times$ fewer tokens than rStar. This may be because search-space control and terminal control concentrate computation on useful exploration, avoiding redundant expansions and extra verification steps. Overall, ASE-MCTS provides an efficient self-evaluated MCTS framework in both latency and accuracy-token trade-off.

5 Conclusion

We presented ASE-MCTS, a fully self-evaluated MCTS framework that improves LLM reasoning without external verifiers, process reward models, or stronger judges. ASE-MCTS combines flexible search-space control, search-oriented reward composition, and dependency-aware asynchronous execution to make noisy self-assessment useful for tree search while reducing execution latency.

Experiments on mathematical and scientific reasoning benchmarks show that ASE-MCTS improves performance especially when correct trajectories are sparse. Signal analysis and ablations confirm the complementary roles of rollout, process, and pairwise rewards, while efficiency results show favorable accuracy-token trade-offs and sub-

stantial latency reduction. These findings support fully self-evaluated MCTS as a practical test-time search strategy for hard reasoning.

Limitations

Although ASE-MCTS improves fully self-evaluated tree search, several limitations remain. First, the current self-assessment value uses a fixed linear combination of rollout, process, and pairwise signals. Although the tested configuration yields a modest improvement in our end-to-end ablation, the gain from reward composition is relatively small and may be sensitive to stochastic self-evaluation, model choice, and weight settings. We therefore view the current aggregation as a simple implementation of complementary search signals rather than a universally robust or optimal fusion rule. Future work should examine variance across repeated runs and develop adaptive or uncertainty-aware strategies for combining these signals.

Second, step-wise tree search may disrupt the natural coherence of reasoning trajectories. Since ASE-MCTS searches over intermediate states, continuous reasoning must be segmented into expandable steps, which may weaken long-range dependencies or make locally plausible states difficult to continue coherently.

Third, our experiments mainly focus on mathematical and scientific reasoning, where final answers are relatively clear and intermediate steps are naturally structured. Extending ASE-MCTS to broader settings, such as tool use, planning, or open-ended decision-making, may require more careful action-space control to balance flexible exploration with valid and task-relevant reasoning.

Acknowledgments

This work is supported by the Natural Science Foundation of China under Grant Nos. 72225011 and 72434005.

References

AI-MO. 2024a. [Aime 2024](#).

AI-MO. 2024b. [Amc 2023](#).

Zhengyu Chen, Yudong Wang, Teng Xiao, Ruochen Zhou, Xuesheng Yang, Wei Wang, Zhifang Sui, and Jingang Wang. 2025. From mathematical reasoning to code: Generalization of process reward models in test-time scaling. *arXiv preprint arXiv:2506.00027*.

Ziru Chen, Michael White, Ray Mooney, Ali Payani, Yu Su, and Huan Sun. 2024. When is tree search useful for llm planning? it depends on the discriminator. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13659–13678.

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

Yifu Ding, Wentao Jiang, Shunyu Liu, Yongcheng Jing, Jinyang Guo, Yingjie Wang, Jing Zhang, Zeng-mao Wang, Ziwei Liu, Bo Du, Xianglong Liu, and Dacheng Tao. 2025. [Dynamic parallel tree search for efficient LLM reasoning](#). In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11233–11252, Vienna, Austria. Association for Computational Linguistics.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. [The llama 3 herd of models](#). *Preprint*, arXiv:2407.21783.

Xinyu Guan, Li Lyna Zhang, Yifei Liu, Ning Shang, Youran Sun, Yi Zhu, Fan Yang, and Mao Yang. 2025. rstar-math: Small llms can master math reasoning with self-evolved deep thinking. In *Forty-second International Conference on Machine Learning*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 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.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, et al. 2024. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3828–3850.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

- Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2024. Large language models cannot self-correct reasoning yet. In *The Twelfth International Conference on Learning Representations*.
- Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pages 611–626.
- Peiji Li, Kai Lv, Yunfan Shao, Yichuan Ma, Linyang Li, Xiaoqing Zheng, Xipeng Qiu, and Qipeng Guo. 2025. [FastMCTS: A simple sampling strategy for data synthesis](#). In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 24405–24422, Vienna, Austria. Association for Computational Linguistics.
- 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 *The Twelfth International Conference on Learning Representations*.
- Anji Liu, Jianshu Chen, Mingze Yu, Yu Zhai, Xuewen Zhou, and Ji Liu. 2020. Watch the unobserved: A simple approach to parallelizing monte carlo tree search. In *International Conference on Learning Representations*.
- Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegrefe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2023. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36:46534–46594.
- Kou Misaki, Yuichi Inoue, Yuki Imajuku, So Kuroki, Taishi Nakamura, and Takuya Akiba. 2025. Wider or deeper? scaling llm inference-time compute with adaptive branching tree search. In *ICLR 2025 Workshop on Foundation Models in the Wild*.
- OpenAI. 2024. Openai o1 system card. <https://openai.com/index/openai-o1-system-card/>. Accessed: 2024-12-05.
- Zhenting Qi, Mingyuan Ma, Jiahang Xu, Li Lyna Zhang, Fan Yang, and Mao Yang. 2025. [Mutual reasoning makes smaller LLMs stronger problem-solver](#). In *The Thirteenth International Conference on Learning Representations*.
- David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. 2024. Gpqa: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*.
- Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2025. [Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning](#). In *The Thirteenth International Conference on Learning Representations*.
- Ye Tian, Baolin Peng, Linfeng Song, Lifeng Jin, Dian Yu, Lei Han, Haitao Mi, and Dong Yu. 2024. Toward self-improvement of llms via imagination, searching, and criticizing. *Advances in Neural Information Processing Systems*, 37:52723–52748.
- Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837.
- Mengsong Wu, Di Zhang, Yuqiang Li, Dongzhan Zhou, and Wenliang Chen. 2025. Selt: Self-evaluation tree search for llms with task decomposition. *arXiv preprint arXiv:2506.07557*.
- An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.
- An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, and 23 others. 2024. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.
- Jinwei Yao, Kexun Zhang, Kaiqi Chen, Jiaxuan You, Zeke Wang, Binhang Yuan, and Tao Lin. 2024. Deft: Flash tree-attention with io-awareness for efficient tree-search-based llm inference. In *ICLR 2024 Workshop: How Far Are We From AGI*.
- 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.
- Di Zhang, Jianbo Wu, Jingdi Lei, Tong Che, Jiatong Li, Tong Xie, Xiaoshui Huang, Shufei Zhang, Marco Pavone, Yuqiang Li, Wanli Ouyang, and Dongzhan Zhou. 2025. [LLaMA-berry: Pairwise optimization for olympiad-level mathematical reasoning via o1-like Monte Carlo tree search](#). In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 7315–7337, Albuquerque, New Mexico. Association for Computational Linguistics.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623.

A Reproducibility Details

A.1 Experimental Setup

We evaluate on challenging mathematical and scientific reasoning benchmarks that leave sufficient headroom for test-time search. We use MATH500, AMC2023, and AIME2024 for mathematical reasoning, and GPQA-Diamond and Olympiad-Physics for scientific reasoning. Easier benchmarks commonly used in prior MCTS work were excluded because the evaluated backbone models showed near-saturated or weakly discriminative performance in our preliminary experiments.

We use the official test or evaluation splits: 500 problems from MATH500, 40 from AMC2023, 30 from AIME2024, 198 from GPQA-Diamond, and 236 from Olympiad-Physics. All generation-based experiments use the same vLLM backend.

A.2 Baseline Configuration and Compute Budgets

Tree-search methods differ in their node definitions, expansion strategies, and evaluation procedures, making exact token- or wall-clock-matched comparisons difficult. We therefore report accuracy together with token consumption and inference time.

For Tree-of-Thoughts (ToT), each node is expanded into five candidates, each candidate is evaluated five times with the scores averaged, the top 10 nodes are retained at each depth, and the maximum depth is 10. On open-ended reasoning tasks, ToT may continue expanding intermediate states without an explicit termination signal, resulting in relatively high generation cost.

For rStar, we use the authors’ released implementation, which generates slightly more than 100 nodes per problem and requires approximately 180 LLM calls on average in our setting.

In the *Latency Reduction* experiment, we configure ASE-MCTS to use approximately 180 LLM calls to maintain a comparable invocation scale to rStar. Here, multiple generations issued from the same prompt in a single batched vLLM request are counted as one LLM call.

The synchronous MCTS baseline uses the same search procedure, generation settings, and random seeds as ASE-MCTS; the only difference is that LLM operations are executed synchronously rather than with our asynchronous scheduling mechanism.

A.3 Answer Verification

Exact string matching can underestimate accuracy on complex reasoning benchmarks because semantically equivalent answers may differ in formatting, units, LaTeX notation, numerical representation, or ordering. We therefore use a three-stage verification pipeline.

First, deterministic rules handle directly verifiable cases, including numerical and Boolean answers. Floating-point values are compared with a tolerance of 10^{-5} , together with task-specific normalization where applicable. Answers that remain unresolved after this stage are passed to two independent LLM judges, **DeepSeek-v3.2** and **Qwen3-Max**, which determine whether the predicted answer is semantically equivalent to the reference answer while ignoring superficial formatting differences. Finally, all cases accepted as correct by the LLM-based verification stage are manually reviewed to remove potential false positives.

Thus, the verification stages are applied sequentially: deterministic matching resolves straightforward cases, LLM judging recovers semantically correct answers missed by rule-based matching, and human review validates the additional cases accepted by the LLM judges. Compared with strict answer matching, this verification procedure increases the measured accuracy of both ASE-MCTS and the evaluated baselines by recovering valid answers that differ only in representation.

A.4 Difficulty Partitioning

In this work, We partition GPQA-Diamond and Olympiad-Physics by empirical base-model difficulty (Section 4.2). For each problem, we estimate its pass rate using 64 independent samples from the corresponding backbone model and sort the problems into four quartiles, Q1–Q4, from easiest to hardest. We then compare ASE-MCTS and Self-Consistency within each quartile. This partitioning is used only for analysis and does not affect inference.

B ASE-MCTS Implementation Details

B.1 MCTS Backbone

ASE-MCTS follows the standard MCTS procedure of selection, expansion, evaluation, and backup. Each node represents a partial reasoning state. During selection, the search recursively chooses child nodes using a standard UCT criterion that balances the current value estimate with an exploration bonus based on visit counts.

Selected nodes are expanded using the search-space control mechanism described in Section 3.1. For the initial expansion from the root, we use high-temperature repeated sampling to generate diverse starting trajectories. For subsequent expansions, ASE-MCTS adopts a hierarchical candidate-proposal mechanism: the model first proposes several distinct candidate directions for continuing the current reasoning state, and each proposed direction is then separately expanded into a complete child node. This two-stage expansion encourages the tree to explore qualitatively different solution paths rather than repeatedly sampling minor variations of the same continuation.

Newly generated states are evaluated using the self-evaluated reward $Q(s)$ defined in Section 3.2, which combines rollout, process, and pairwise signals. The resulting value is backed up along the selected path, where ancestor visit counts are incremented and value estimates are updated by averaging accumulated evaluations.

The search continues until the iteration budget is reached, while semantic terminal detection prevents further expansion of completed reasoning states. ASE-MCTS retains this standard MCTS backbone and mainly modifies candidate generation, self-evaluation, and the scheduling of LLM calls.

B.2 Reward Computation

Pairwise reward provides a local preference signal among sibling nodes. When a node is expanded into a sibling set $\mathcal{S} = \{s_1, \dots, s_m\}$, we perform pairwise comparisons over all sibling pairs under the same parent context. Each comparison prompt contains the input problem, the shared reasoning history, and two candidate next states. The evaluator outputs a signed preference margin

$$v_{ij} \in [-5, 5],$$

where a positive value favors s_i over s_j , a negative value favors s_j , and values close to zero indicate

no clear preference.

We first normalize each comparison score to $[0, 1]$:

$$\tilde{v}_{ij} = \frac{v_{ij} + 5}{10}.$$

The pairwise reward of each sibling is then obtained using a Borda-style aggregation over all of its pairwise comparisons:

$$V_{\text{pair}}(s_i) = \frac{1}{m-1} \sum_{j \neq i} \tilde{v}_{ij}.$$

Thus, $V_{\text{pair}}(s_i) \in [0, 1]$ summarizes how strongly s_i is preferred relative to the other newly expanded sibling nodes. Since these comparisons are restricted to candidates sharing the same parent, the pairwise reward serves primarily as a local discrimination signal and is combined with the rollout and process rewards for node evaluation.

B.3 Search and Aggregation Configuration

ASE-MCTS uses a fixed search configuration across the main experiments unless otherwise specified. We run 15 MCTS iterations with an exploration constant of 12, an initial branch factor of 4, and a branch factor of 3 for subsequent expansions. Each evaluated node uses 3 rollout trajectories, and the maximum search depth is 10. The initial children are generated by repeated sampling with temperature 0.9, while subsequent expansions use the hierarchical candidate-proposal mechanism described in Section B.1.

For node evaluation, the rollout, process, and pairwise signals are combined as

$$Q(s) = 0.4V_{\text{roll}}(s) + 0.5V_{\text{proc}}(s) + 0.1V_{\text{pair}}(s).$$

where the three reward signals are normalized to a common scale before fusion. The rollout and process rewards provide the primary value estimates, while the pairwise component supplies complementary local discrimination among sibling states.

After MCTS terminates, ASE-MCTS performs an additional aggregation stage to select the final answer from the candidate solutions represented by the terminal leaves. For each candidate leaf l , we consider three complementary signals: its consistency with other candidate solutions, its own leaf value, and the average value along its reasoning path. The final aggregation score is

$$S(l) = 0.4C(l) + 0.3Q(l) + 0.3\bar{Q}_{\text{path}}(l),$$

where $C(l)$ denotes the consistency of the answer associated with l among the candidate solution set, and

$$\bar{Q}_{\text{path}}(l) = \frac{1}{|\mathcal{P}(l)|} \sum_{s \in \mathcal{P}(l)} Q(s)$$

is the average node value along the reasoning path $\mathcal{P}(l)$. The candidate with the highest aggregation score is returned as the final solution.

C Additional Analysis

C.1 Extended Reward Analysis

For the reward analysis, we sampled approximately 80 cases from the benchmarks used in our main experiments and collected about 210 intermediate nodes, organized as sibling groups of three. For each node, we recorded the reward signals produced by ASE-MCTS and independently obtained human quality ratings. We further asked several strong LLMs, including DeepSeek, GPT, Qwen, and DouBao, to judge the same nodes. Their scores were averaged to construct the ensemble target (ET), which serves as a consensus-based reference for analyzing the behavior of different reward signals.

Figure 7 reports the extended correlation matrix, including individual LLM judges and additional reward variants considered during development. In particular, rollout_SC measures answer consistency among rollout trajectories, while rollout_confidence uses the confidence of the generated rollout trajectories; both show relatively weak agreement with human and ensemble judgments and were therefore not adopted in the final reward. Two broader patterns are visible. First, the strong LLM judges exhibit substantial mutual agreement, while their averaged ET score shows stronger agreement with human ratings than the individual judges. Second, combining the rollout, process, and pairwise signals improves agreement with both ET and human ratings compared with the individual reward signals, providing additional support for the reward fusion used in ASE-MCTS.

C.2 Additional Analysis of rStar

We provide additional analysis of rStar to better contextualize its behavior in our reproduction. Two aspects are particularly relevant: the repeated-sampling components within its generator and the capability relationship between the generator and discriminator.

First, the rStar generator itself incorporates substantial self-consistency-style sampling and voting. In the original method, terminal-node rewards are computed from self-consistency majority-vote confidence, and the authors additionally report rStar(generator @maj) to isolate the generator from the discriminator. The released implementation also shows that several actions invoke multiple samples during expansion or evaluation. For example, the direct-answer action A2 and terminal subquestions in A3 use mcts_num_last_votes, whose default value is 32, while A4 aggregates multiple re-answers into a value estimate. Thus, rStar combines MCTS-based exploration with local repeated sampling and voting, which is important context when comparing it with self-consistency baselines.

Second, the effectiveness of mutual-consistency discrimination appears to depend on the capability relationship between the generator and discriminator. The discriminator ablation in the original paper shows that different discriminator choices yield different gains. With LLaMA3-8B-Instruct as the generator, majority voting achieves 88.70, while using LLaMA3-8B-Instruct, LLaMA3.1-8B-Instruct, Phi3-Mini-Instruct, and GPT-4 as discriminators gives 88.78, 89.52, 91.13, and 92.57, respectively. This suggests that the quality of the verification signal can vary with the discriminator.

Our reproduction shows a related pattern on more challenging mathematical and scientific reasoning benchmarks. As shown in Table 1, rStar(generator @maj) performs below SC@maj32 in our evaluated settings. Applying the rStar discriminator slightly improves the Llama3.1-8B-Instruct setting, but decreases performance for Qwen2.5-7B-Instruct and Qwen3-8B relative to the generator-only variant. This suggests that mutual-consistency verification is not capability-neutral: when the discriminator is less capable than the generator on the target task, its verification signal may become unreliable and can adversely affect trajectory selection. This dependence is plausible because rStar asks the discriminator to continue partial trajectories produced by the generator and assess consistency with the candidate answer; a less capable discriminator may fail to complete an otherwise valid trajectory or may reinforce an incorrect one.

These observations do not imply that the rStar discriminator is ineffective in general. Rather, its benefit should be interpreted under the specific generator-discriminator pairing and task setting.

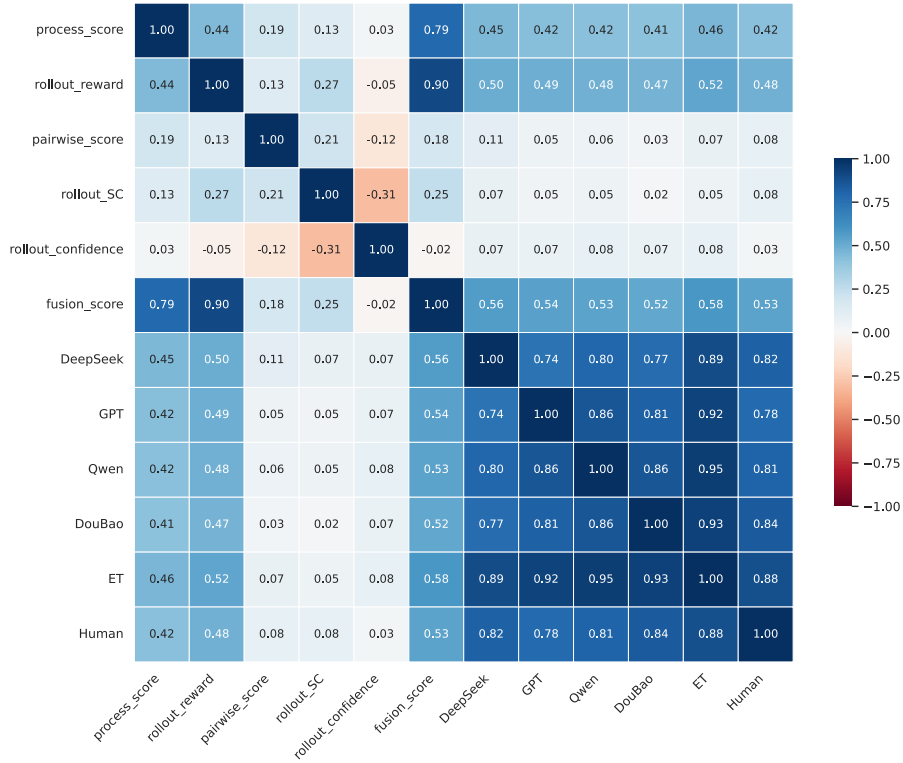


Figure 7: Extended Pearson correlation analysis among reward signals, human ratings, and strong-LLM judges.

ASE-MCTS considers a complementary setting in which search guidance is derived entirely from the generator itself, without assuming access to an additional discriminator of comparable or greater task capability.

C.3 Applicability to Larger Models

ASE-MCTS is architecturally scale-agnostic and can be applied to stronger foundation models. In the main experiments, we focus on small and medium-sized models because their noisy self-assessment provides a more challenging testbed for fully self-evaluated MCTS. To further examine whether ASE-MCTS remains effective beyond this setting, we conduct additional experiments with DeepSeek-V3 through API access.

Method	GPQA-Diamond
Zero-shot CoT	63.13
Best-of-N@16	64.14
Self-Consistency@maj32	65.15
ASE-MCTS	66.16

Table 6: Additional results on DeepSeek-V3.

As shown in Table 6, ASE-MCTS still improves over strong sampling baselines on GPQA-Diamond. The gain is smaller than that observed on 7-8B

models, which is expected because stronger foundation models already have much higher single-pass reasoning accuracy and more reliable self-assessment. Nevertheless, the result suggests that ASE-MCTS remains applicable to larger models, while its relative advantage is more pronounced in the small- and medium-model regime where correct trajectories are sparser and self-evaluation is noisier.

D Prompt and Search-Guide Details

D.1 Prompt Templates

ASE-MCTS uses eight prompt templates for node expansion, rollout generation, self-evaluation, pairwise comparison, and final answer generation. Specifically, the templates include *Initial Expand*, *Candidate Proposal*, *Idea Expand*, *Fast Rollout*, *Process Evaluation*, *Reward Evaluation*, *Pair Evaluation*, and *Best Answer*. Their exact contents are provided in Figures 8–15.

All prompts are instantiated through a structured chat template with separate fields for the current reasoning path, system instructions, task-specific input, and assistant output prefix. To improve KV-cache reuse across calls sharing the same search state, we serialize the node path as a shared prefix before the system and user instructions. The

assistant prefix is used to constrain the beginning of model outputs to predefined structures such as <step> or <ideas>, facilitating stable downstream parsing.

D.2 Search Guide

The search guide provides dataset-specific instructions that instantiate the corresponding placeholders in the prompt templates. It allows the same MCTS procedure to adapt its expansion and evaluation criteria to different reasoning domains. Each search guide contains six components:

Idea Guidance: Directs the high-level movement of the search by providing constraints on the proposal of new core ideas. It helps the model decide when to explore further or when to converge on a final answer.

- *AMC2023 (Math): “If the conditions are sufficient, consider requesting to finish the solution and return the final answer.”*
- *GPQA-Diamond (Science): “Consider multiple steps to solve the problem.”*

Expand Guidance: Imposes fine-grained constraints on the reasoning process during node expansion, such as requiring step-by-step verification or prohibiting specific output types like code.

- *AMC2023 (Math): “Perform calculations step-by-step to avoid arithmetic errors. If using a theorem, verify its conditions first. Double-check algebraic signs and unit consistency. Don’t give any code.”*
- *GPQA-Diamond (Science): “If the conditions are sufficient, consider requesting to finish the solution and return the final answer.”*

Process Criteria: Defines the evaluation standard for individual reasoning steps. These are used by the self-critic to detect hallucinations, logical gaps, or violations of problem constraints.

- *AMC2023 (Math): “1. Scrutinize for arithmetic errors or misuse of theorems. 2. Check if the step ignores problem constraints (e.g., ‘distinct integers’).”*
- *GPQA-Diamond (Science): “1. Identify and penalize factual hallucinations or formula misuse. 2. Detect logical disconnects relative to the scientific context.”*

Pairwise Criteria: Sets the comparative standard for the pair_evaluation prompt, focusing on identifying relative weaknesses such as ambiguity or failure to consider edge cases.

- *AMC2023 (Math): “1. Which solution exhibits more calculation errors? 2. Identify which solution fails to consider edge cases (e.g., division by zero).”*
- *GPQA-Diamond (Science): “1. Which solution contains more factual inaccuracies? 2. Identify which solution relies on weaker assumptions.”*

Reward Objectives: Establishes the global scoring logic for full trajectories, emphasizing numerical accuracy for math and adherence to domain-specific laws for science.

- *AMC2023 (Math): “1. Incorrect final answer: Score must be 0 or extremely low. 2. Penalize logical leaps or hallucinated facts even if the answer is correct by chance.”*
- *GPQA-Diamond (Science): “1. Severe penalty if the final option contradicts the derived reasoning. 2. Scrutinize for violations of domain-specific scientific laws.”*

Answer Restrictions: Defines the strict formatting requirements for the final output to ensure automated evaluation can parse the answer.

- *AMC2023 (Math): “1. The answer must be in the format of a number. 2. The answer must be a valid number.”*
- *GPQA-Diamond (Science): “1. The answer must be a choice from A, B, C, D. 2. If no perfect match, select the most reasonable one.”*

E Use of AI Assistants

AI assistants were used for language polishing, code editing and debugging support, figure refinement, and manuscript preparation. All scientific claims, experimental designs, implementation decisions, results, analyses, and final content were independently verified by the authors.

F Artifact Licenses and Use.

We use publicly available benchmarks and models for research evaluation purposes. MATH500,

Initial Expand Prompt

<|system|>

A conversation between the User and the Assistant. The user asks a question that requires a **step-by-step** solution. The Assistant provides **only the first step**, including the **core idea** of this step and then its **full content**.

[init_expand_guidance]

The step must be marked with **<step> </step>** at the beginning and end.

Format requirement:

<step>

[Core idea]: xxx

[Full step]: xxxxxx

</step>

<|user|>

User question: [user question]

<|assistant|>

Answer: <step>\n[Core idea]:

Figure 8: Initial-Expand prompt template.

AMC2023, AIME2024, GPQA-Diamond, and Olympiad-Physics are used only for benchmark evaluation. The open-source models and software packages used in our experiments are used under their respective licenses or access conditions. We do not redistribute the original datasets or model weights.

Candidate Proposal Prompt

<|system|>

<|user|>

Given the context above, provide **[num_ideas]** distinct and diverse core ideas for the next reasoning step. Make sure these ideas explore different approaches or directions. **[idea guidance]** Note that we only need concise core ideas without full steps. Format each idea as [1]: idea1, [2]: idea2, etc. </ideas>\n Be concise but clear. The ideas must be marked with **<ideas>** </ideas> at the beginning and end.

<|assistant|>

[previous steps] **[num_ideas]**

Here are distinct approaches for the next step:

<ideas>

Figure 9: Candidate-Proposal prompt template..

Idea Expand Prompt

[previous steps]

<|system|>

You must output the following three sections in order:

1. **[Core idea]** - The core concept or key insight for this step
2. **[Full step]** - The complete implementation details
3. **[Check terminal]** - Terminal check result (output format: 'The final answer to the question is **\\boxed{answer}**' or 'Not a terminal node yet.')

<|user|>

Follow the existing steps and provide the core idea and full content of the next step (one step only). **[expand guidance]**

First, generate the complete **[Full step]** content. Then, output **[Check terminal]**. If a clear answer is found, strictly output only 'The final answer to the question is **\\boxed{answer}**' without any additional information or content. If the final answer has not been reached yet, output 'Not a terminal node yet.'

The step must be marked with **<step>** </step> at the beginning and end.

<|assistant|>

The next step:

<step>

[Core idea]: [core instruction]

[Full step]:

Figure 10: Idea-Expand prompt template.

Fast Rollout Prompt

[previous steps]

<|system|>

<|user|>

Follow the **existing steps** and answer the question till the end, provide the final answer in the last step. Strictly follow the format requirements and put the final answer in **\\boxed{answer}**, then output **</step>**.

<|assistant|>

The rest of the answer:

<step>

Figure 11: Fast-Rollout prompt template.

Process Evaluation Prompt

[previous steps]

<|system|>

<|user|>

Please evaluate the quality of the current step based on the process criterions ([**process criterions**]), and then give a score from 0 to 10.

****Scoring Standard:****

0 means the step is riddled with critical errors, hallucinations, or logic gaps.

5 means the step contains noticeable flaws or overlooks constraints.

10 means no errors, violations, or potential risks can be detected.

Strictly provide your evaluation in the following format:

<evaluation>

[Analysis]: Evaluate and briefly analyze how well the solution meets the objectives.

[Score]: X (where X is a number from 0 to 10)

</evaluation>

The current step is: **[current step]**

<|assistant|>

<evaluation>

[Analysis]:

Figure 12: Process-Evaluation prompt template.

Reward Evaluation Prompt

[full solution]

<|system|>

<|user|>

Please evaluate the quality of the above solution based on the objectives of solving the problem (**[objectives]**), and then give a score from 0 to 10.

****Scoring Standard:****

0 means the solution is riddled with critical errors.

5 means the solution contains noticeable flaws.

10 means no errors, violations, or potential risks can be detected.

Strictly provide your evaluation in the following format:

<evaluation>

[Analysis]: Evaluate and briefly analyze how well the solution meets the objectives

[Score]: X (where X is a number from 0 to 10)

</evaluation>

<|assistant|>

<evaluation>

[Analysis]:

Figure 13: Reward-Evaluation prompt template.

Pair Evaluation Prompt

[previous steps]

<|system|>

<|user|>

Please compare the quality of Step A and Step B based on the process criteria ([pairwise criteria]).

First, conduct an in-depth analysis of their flaws and potential risks, and then evaluate which is better (A, B, or Tied). Give a Score that reflects the difference in error density.

****Scoring Standard:****

0 means both steps have a similar amount of errors or flaws (Tied).

5 means the better step is significantly more rigorous and contains far fewer errors or risks than the other.

Strictly provide your comparison result in the following format:

<comparison>

[Analysis]: Compare and briefly analyze the performance of Step A and Step B in meeting the process criteria respectively, focus on pointing out the advantages and disadvantages of Step A relative to Step B.

[The better step]: A or B or Tied.

[Score]: X (where X is an integer from 0 to 5)

</comparison>

The Step A is: [Step A]

The Step B is: [Step B]

<|assistant|>

<comparison>

[Analysis] :

Figure 14: Pair-Evaluation prompt template.

Best Answer Prompt

<|system|>

<|user|>

[best solution] Please provide your final answer based on the complete solution outlined above. If the above ideas haven't yet led to a final solution, please provide your final answer directly.

[answer restrictions] Please strictly follow the formatting requirements and place your final answer in the **\\boxed{...}**.

<|assistant|>

The final answer is: **\\boxed{**

Figure 15: Best-Answer prompt template.