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

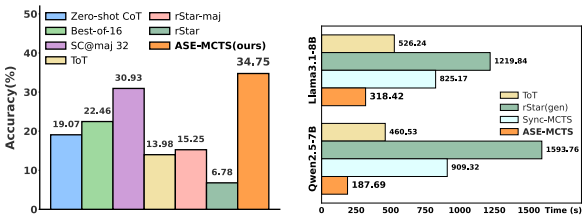
Anonymous ACL submission

Abstract

While Monte Carlo Tree Search (MCTS) effectively enhances LLM reasoning, its widespread application is constrained by the unreliability of intermediate rewards and prohibitive time costs. Prevailing methods typically secure high-quality signals via external supervision, while intrinsic self-evaluation often suffers from noise and unreliability, limiting the scalability and broad applicability of tree search. To address these challenges, we propose ASE-MCTS, an **A**synchronous **S**elf-**E**valuated **MCTS** framework. To improve reward robustness, we introduce a fused reward mechanism that integrates diverse signals, including pairwise comparisons, to enhance both global alignment and local discriminability. Simultaneously, we utilize a novel asynchronous orchestrator that parallelizes independent expansion and simulation tasks, effectively masking generation latency. Extensive experiments demonstrate that ASE-MCTS consistently outperforms strong baselines and achieves capability breakthroughs on high-difficulty benchmarks without external supervision. Moreover, our asynchronous design significantly improves computational efficiency, yielding up to a 6× speedup over synchronous implementations.

1 Introduction

While Large Language Models (LLMs) excel across various domains, they still struggle with complex, multi-step tasks. Although Large Reasoning Models (LRMs) like OpenAI-o1 (OpenAI, 2024) enhance problem-solving via internalized chain-of-thought (Wei et al., 2022), a single sampling pass inherently limits their exploration of the solution space. Inference-time strategies address this limitation: methods like Self-Consistency (Wang et al., 2023) broaden coverage through parallel sampling, while tree-based approaches like Tree of Thoughts (Yao et al., 2023) enable directional search. Notably, Monte Carlo Tree Search (MCTS) (Hao et al., 2023) distinguishes itself by using lookahead simulations to evaluate intermediate steps, efficiently guiding the model through the search space.



(a) Performance Comparison (b) Time Efficiency

Figure 1: (a) Our ASE-MCTS outperforms strong baselines on mathematical and scientific reasoning benchmarks without using any external supervision. (b) The proposed asynchronous engine significantly reduces the time cost of MCTS.

Crucially, MCTS efficacy hinges on the accuracy of intermediate reward signals (Guo et al., 2025). While prevailing approaches secure high-quality signals via external supervision—such as Process Reward Models (PRMs) (Chen et al., 2025; Lightman et al., 2023) or superior LLMs (Tian et al., 2024)—these methods incur prohibitive VRAM and latency overheads. By contrast, self-evaluated MCTS—where the policy model itself guides the search—emerges as a vital solution for ensuring scalability and practical utility.

However, the primary obstacle to effective self-evaluated MCTS lies in the inherent unreliability of LLM self-assessment (Chen et al., 2024). Research indicates that LLMs are frequently biased and inconsistent when evaluating their own generations (Huang et al., 2024; Zheng et al., 2023). This deficiency is particularly acute in Small Language Models (SLMs, e.g., 7-8B), where limited capacity leads to high-variance, noisy feedback that prevents MCTS from effectively selecting the most promising paths for exploration.

Beyond reward reliability, another critical barrier

to test-time tree search is the prohibitive computational cost of extensive LLM generations (Snell et al., 2025). In MCTS, this bottleneck is exacerbated by task duration heterogeneity: lengthy reasoning rollouts significantly outlast expansion and evaluation steps. In traditional serial frameworks, this mismatch causes severe blocking, rendering the approach impractical for time-sensitive applications.

To address these twin challenges, we propose **ASE-MCTS**, an **A**synchronous **S**elf-**E**valuated **MCTS** framework. To enhance reward reliability, we argue that effective signals must balance global value estimation with local discriminability among child nodes. Based on extensive analysis, we design a robust reward fusion mechanism that incorporates a novel pairwise signal to sharpen this local distinction. To improve efficiency, we re-architect the MCTS workflow by modeling the intricate dependencies between expansion, simulation, and evaluation as a Directed Acyclic Graph (DAG), employing a custom asynchronous orchestrator to maximize task parallelism. Consequently, our method achieves superior performance while accelerating inference by several times, significantly expanding the practical applicability of tree search. In summary, our main contributions are as follows:

- Propose **ASE-MCTS**, a fully self-evaluated MCTS framework without any external supervision, demonstrating superior performance on challenging reasoning benchmarks, effectively breaking the capability ceilings of base models on high-difficulty problems.
- Introduce a fused reward mechanism that integrates rollout, process, and novel pairwise signals, providing a robust MCTS reward signal with both **global alignment** and **local discriminability**.
- Re-architect the MCTS workflow as a DAG managed by an **asynchronous orchestrator**. This design maximizes task parallelism, effectively masking the latency of extensive reasoning rollouts and significantly accelerating inference.

2 Related Work

LLM Reasoning and Search Enhancing LLM reasoning capabilities primarily hinges on decomposing 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 Methodology

This section details the methodology of ASE-MCTS. We first outline the framework architecture in Section 3.1. Then, Section 3.2 introduces our fused evaluation strategy and the novel pairwise reward mechanism. Finally, Section 3.3 presents the asynchronous orchestrator designed for computational efficiency.

3.1 ASE-MCTS Framework

We formulate LLM reasoning as a search over a tree $\mathcal{T}$, where nodes s_t represent reasoning steps and leaves correspond to complete solutions. The

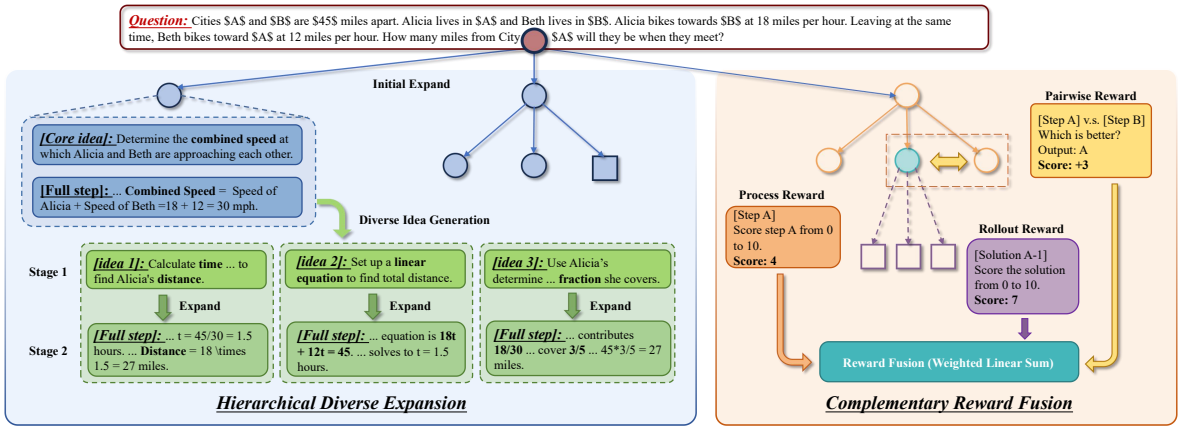


Figure 2: An Overview of the Extension and Simulation Phases of the ASE-MCTS Framework. The left side illustrates the Hierarchical Diverse Expansion mechanism, which enables the LLM to explore a more diverse set of solution approaches. The right side presents the definitions and fusion forms of three types of reward signals, namely rollout reward, process reward, and pairwise reward.

search phase follows the standard MCTS lifecycle—Selection, Expansion, Simulation, and Backpropagation—adapting this framework to LLMs introduces specific challenges in action granularity and diversity. To address these, we introduce two targeted design strategies:

Prompt-Guided Action Space Requirements for action spaces and search objectives vary significantly across reasoning tasks, and rigid or inappropriate action granularity can severely hamper model performance. To address this, we introduce a "Search Guide" module that enables injecting natural language constraints and objectives into expansion and reward prompts. By decoupling task-specific logic from the core algorithm, we achieve architectural universality and adaptive granularity without code modification.

Hierarchical Diverse Expansion Another potential pitfall in tree search for LLMs is "diversity degradation." As search depth increases and context accumulates, simple high-temperature sampling often yields semantically identical paths, leading to redundant exploration. To counteract this, we propose a two-stage hierarchical expansion strategy (Figure 2). We structure each reasoning step into a "Core Idea" and a "Full Step." During expansion, we first prompt the LLM to generate multiple distinct high-level core ideas, which are then independently expanded into full steps containing detailed reasoning. This mechanism enforces diversity at the strategic level, mitigating mode collapse in deep search. Note that for the initial root expansion, we retain standard repeated sampling to prevent initial bias and ensure broad coverage of the solution space.

3.2 Reward Signals Design

With the basic MCTS framework established, fundamental search performance improvements necessitate optimizing reward signal design.

The Dual Role of Reward In the MCTS lifecycle, the reward signal serves to estimate node value, guiding the *Selection* phase to iteratively prioritize promising branches via the Upper Confidence Bound for Trees (UCT) algorithm (Kocsis and Szepesvári, 2006). Consequently, a robust mechanism must satisfy two orthogonal objectives: **Global Alignment**, ensuring signal magnitude correlates with objective solution quality, and **Local Discriminability**, ensuring the effective ranking of sibling nodes to prioritize exploration.

Signal Definitions To achieve this balance, we design three distinct self-evaluated reward signals. While rollout and process rewards provide standard value estimation, we introduce a novel pairwise reward (Guan et al., 2025; Zhang et al., 2025) to explicitly enhance local discrimination.

- **Rollout Reward** (V_{roll})**:** Leverage lookahead simulations to estimate the future value of the current state. We sample N paths $\{y_1, ..., y_N\}$ from state s_t, scoring their final quality on

$[0, 10]$ to calculate the expected value:

$$V_{roll}(s_t) = \frac{1}{N} \sum_{k=1}^{N} \text{LLM}(\text{prompt}_{roll}, y_k) \quad (1)$$

- **Process Reward (V_{proc}):** Assess the immediate logical validity of step s_t given history h_{t-1} without lookahead, yielding a scalar score in $[0, 10]$:

$$V_{proc}(s_t) = \text{LLM}(\text{prompt}_{proc}, h_{t-1}, s_t) \quad (2)$$

- **Pairwise Reward (V_{pair}):** This signal provides the relative advantages among child nodes. For a set of sibling nodes $\mathcal{S}_{sib} = \{s^1, s^2, ..., s^m\}$, we form pairs (s^i, s^j) and prompt the model to judge which step is better given the context, assigning a "margin of victory" score $v_{ij} \in [-5, +5]$.

$$v_{ij} = \text{LLM}(\text{prompt}_{pair}, h_{t-1}, s^i, s^j) \quad (3)$$

These margins are aggregated into a bounded node value via a weighted Borda Count (Zhang et al., 2025). The pairwise reward for node s_i is computed as:

$$V_{pair}(s_i) = 10 \times \text{sigmoid}\left(\frac{\sum_{p \in P_i} v_p}{T \cdot M_i}\right) \quad (4)$$

where P_i represents the set of all pairwise comparisons involving s_i, v_p is the signed victory margin, M_i is the number of comparisons performed for s_i. Here, the sigmoid function and temperature T are utilized to map potentially high-noise signals into a smoothed value interval in $[0, 10]$.

Fusion Mechanism We employ a linear fusion strategy to derive the final node value $Q(s)$, integrating global process/rollout rewards with the local refinement of the pairwise signal:

$$Q(s) = \alpha \cdot V_{roll}(s) + \beta \cdot V_{proc}(s) + \gamma \cdot V_{pair}(s) \quad (5)$$

We set $\alpha, \beta \gg \gamma$ to prioritize global consistency, utilizing γ specifically for fine-grained tie-breaking among high-quality candidates during selection.

3.3 Asynchronous Orchestrator

Traditional MCTS implementations execute generation and evaluation steps sequentially, creating "blocking" bottlenecks where the GPU remains underutilized. This bottleneck becomes particularly pronounced in ASE-MCTS, given the increased computational load imposed by its additional evaluation mechanisms. However, logic implies that many sub-tasks—such as evaluating independent trajectories—lack causal dependencies and can be parallelized. To exploit this, we re-architect the MCTS workflow as a dynamic directed acyclic graph (DAG). Instead of fixed procedural loops, we conceptualize the search as a set of atomic operations (e.g., Expansion, Rollout, Evaluation) governed strictly by data dependencies. This allows an asynchronous orchestrator to continuously schedule independent tasks into the inference engine without artificial waiting periods.

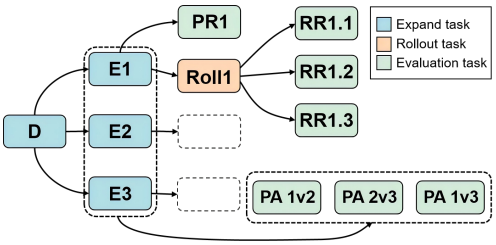


Figure 3: Task dependency graph of the asynchronous orchestrator. Nodes represent atomic LLM generation tasks, while directed edges denote data dependencies and dotted boxes denote batch creation. **D**: Diverse Idea Generation; **E**: Expansion; **Roll**: Rollout Simulation; **PR** (Process Reward); **RR** (Rollout Reward); **PA** (Pairwise Reward).

As shown in Figure 3, the workflow initiates with diverse idea generation, where subsequent expansions and simulations form mutually independent branches. We employ an event-driven mechanism ensuring that successor tasks are instantiated immediately upon the completion of their predecessors. Consequently, all tasks without logical dependencies are executed in parallel, regardless of their branch origin. This design transforms the search into a high-throughput pipeline that maximizes GPU saturation. By leveraging a space-time trade-off—utilizing VRAM to maintain concurrent states—we effectively mask the latency of extensive self-evaluations in ASE-MCTS and significantly compress the total search time.

4 Experiments

Models We evaluate three SLMs representing a hierarchy of increasing capability based on chronological release and empirical perfor-

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
			Llama3.1-8B-Instruct		
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	62.60	35.00	**13.33**	**33.33**	10.17
Self-Consistency@maj32	**64.80**	32.50	10.00	32.83	11.86
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**
			Qwen2.5-7B-Instruct		
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	16.67	35.35	17.80
Self-Consistency@maj16	80.20	62.50	13.33	37.37	18.64
Self-Consistency@maj32	81.80	62.50	16.67	35.86	19.07
ToT(beam search)	62.40	37.50	6.67	32.83	11.69
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**
			Qwen3-8B		
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	77.50	36.67	53.54	27.97
Self-Consistency@maj16	90.20	**80.00**	40.00	55.55	30.51
Self-Consistency@maj32	**91.20**	**80.00**	40.00	56.57	30.93
ToT(beam search)	65.80	42.50	15.00	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	90.60	**80.00**	**43.33**	**58.08**	**34.75**

mance: Llama3.1-8B-Instruct (Dubey et al., 2024), Qwen2.5-7B-Instruct (Qwen et al., 2025), and Qwen3-8B (Yang et al., 2025).

Datasets For mathematical reasoning, we employ MATH500 (Hendrycks et al., 2021), AMC2023 (AI-MO, 2024b), and AIME2024 (AI-MO, 2024a), representing a clear progression from standard benchmarks to high-difficulty competition problems. For scientific reasoning, we integrate GPQA-Diamond (Rein et al., 2024) and Olympiad-Physics (He et al., 2024), evaluating the models' capabilities across both graduate-level multidisciplinary questions and highly complex physical derivation tasks.

Baselines We categorize baselines into parallel sampling and tree search strategies. For sampling, alongside **Zero-shot CoT** (Wei et al., 2022), we select **Self-Consistency (SC)** (Wang et al., 2023) and **Best-of-N** (Cobbe et al., 2021). While both sample N trajectories, they represent distinct aggregation methods: SC relies on statistical consensus via majority voting, whereas Best-of-N utilizes explicit 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. Additionally, we adopt **rStar** (Qi et al., 2025) as a representative MCTS baseline, adhering to its original hyperparameters to evaluate both its generator-majority and discriminator-guided variants.

4.1 Main Results

ASE-MCTS Superiority As illustrated in Table 1, ASE-MCTS consistently achieves leading performance across almost all of tested models and datasets. Notably, the method demonstrates exceptional scalability, amplifying the reasoning capabilities of both weaker (Llama-3.1-8B) and stronger (Qwen-3-8B) foundation models. This advantage

is particularly pronounced on high-difficulty benchmarks like GPQA-Diamond and Olympiad-Physics. While conventional strategies often plateau on these tasks due to the sparsity of correct solutions, our method effectively leverages guided search to navigate complex logical chains, breaking the capability ceilings of the base models.

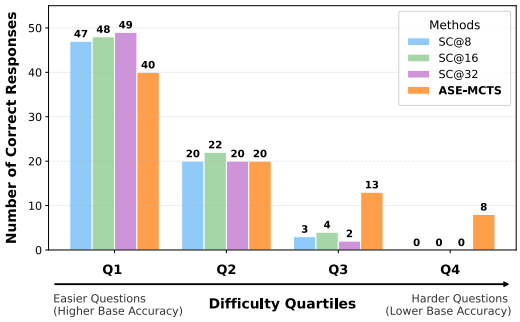
Analysis of Baseline Limitations While SC proves to be a robust baseline, it is inherently limited by the model's zero-shot pass rate (further analyzed in Sec 4.2). Regarding search-based methods, ToT generally underperforms due to the sensitivity of beam search to noisy process rewards. Similarly, rStar exhibits limited gains on these intensive tasks. We find that when the discriminator model is not significantly stronger than the generator, reliance on mutual consistency can be counterproductive, rejecting correct but complex solutions that the discriminator fails to align with. ASE-MCTS overcomes this by integrating multi-view evaluations, ensuring robust guidance even without an external superior judge.

4.2 Performance Analysis by Difficulty

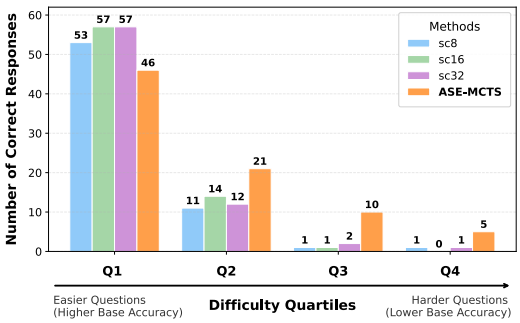
While SC serves as a formidable baseline, its efficacy is inherently bounded by the model's intrinsic probability distribution; it struggles when the correct solution is not dominant in the sampling space. To provide a deeper analysis of ASE-MCTS's effectiveness and its specific advantages over SC, we designed an experiment to strictly evaluate their performance across problems of varying difficulty levels.

Difficulty Partitioning We focused our analysis on the two most challenging datasets, GPQA and Olympiad-Physics. To quantify the intrinsic difficulty of each problem, we calculated the average pass rate based on 64 independent samples from the base model. Based on these empirical scores, the datasets were sorted and partitioned into four distinct quartiles (Q1 to Q4), ranging from Easy to Hard.

Breaking Capability Ceilings We compare the performance of ASE-MCTS against SC (with k=8, 16, 32) across these levels, as illustrated in Figure 4. On lower-difficulty tasks (Q1), ASE-MCTS maintains performance comparable to SC. However, a significant divergence emerges in the medium-to-high difficulty tiers. While SC degrades rapidly as incorrect hallucinations drown out the majority



(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).

vote on hard problems, ASE-MCTS maintains superior stability. By leveraging reward signals to navigate the search space, our method effectively identifies sparse, high-value trajectories, solving complex problems where the base model's zero-shot accuracy approaches 0%.

4.3 Analysis of Reward Signals

To validate the hybrid reward design proposed in Section 3.2, we conduct a quantitative analysis of different reward signals. Specifically, we investigate whether the proposed signals successfully fulfill the dual objectives of **Global Alignment** and **Local Discriminability**.

Ground Truth Construction To evaluate reward quality, we constructed a ground truth set using 200 intermediate nodes sampled from AMC2023. We employed state-of-the-art LLMs (DeepSeek-V3.2, GPT5-mini, Qwen3-Max, Doubao-1.8) alongside human experts to independently annotate node values, analyzing their alignment via Pearson correlations (Figure 5). To mitigate individual model

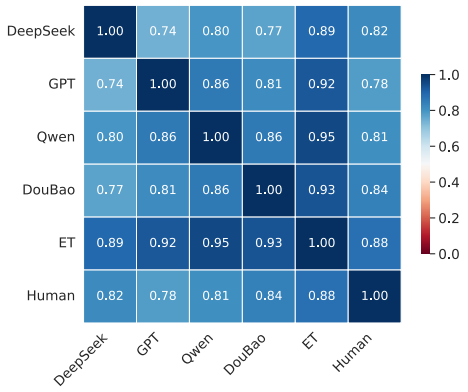


Figure 5: Pearson correlation heatmap of value assessments from different LLMs and human experts. Fused score ET represents Ensemble Truth and achieves the highest 0.88 correlation.

Signal	AMC2023		GPQA-Diamond	
	Pearson(↑)	Regret(↓)	Pearson(↑)	Regret(↓)
		Qwen2.5-7B-Instruct		
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
Roll-SC	0.05	0.3563	0.01	0.3411
Roll-Conf	0.08	0.3117	0.03	0.3246
Fusion	**0.58**	0.2216	**0.40**	0.2846
		Qwen3-8B		
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**
Roll-SC	0.18	0.4527	0.16	0.4416
Roll-Conf	-0.20	0.3937	-0.11	0.4048
Fusion	**0.65**	0.2719	**0.57**	0.2287

Table 2: Signal performance on AMC 2023 and GPQA-Diamond. **Bold** indicates the best result, and underline indicates the second best result. The Fusion demonstrates consistent improvements.

biases, we normalized and aggregated scores into an equal-weighted "Ensemble_Truth" (ET). This fused metric demonstrates high consistency ($r = 0.88$) with human judgments, validating V_{ET} as a robust gold standard.

Evaluation Metrics Subsequently, we sampled 600 nodes from both the AMC2023 and GPQA datasets for testing. To assess global alignment, we calculate the Pearson correlation coefficient between each reward signal and the ensemble truth V_{ET}. For local discriminability, we employ *Regret* ($\mathcal{R}$) to quantify the value loss caused by a signal's ranking error. Given a set of sibling nodes $\mathcal{S}$ and our signal ϕ, let s^* be the node with the highest V_{ET}, and $\hat{s}$ be the node selected by maximizing signal ϕ, the regret is defined as:

$$\mathcal{R}(\phi) = V_{ET}(s^*) - V_{ET}(\hat{s}) \tag{6}$$

A lower $\mathcal{R}$ indicates that the signal ϕ successfully identifies high-quality candidates, minimizing the gap between the selected and the objective optimal node.

Beyond the signals V_{roll}, V_{proc}, and V_{pair} defined in our method, we evaluate two comparative signals: **Rollout Self-Consistency**, which measures the semantic consensus of answers among rollout trajectories, and **Rollout Confidence**, derived from the average log-probabilities of the generated tokens during rollouts.

Results and Insights Consistent with intuition, the reliability of reward signals correlates positively with the base model's capability and negatively with dataset difficulty. Crucially, we observe that V_{roll} and V_{proc} achieve superior global alignment, acting as reliable anchors for the general search direction. Conversely, while V_{pair} exhibits weaker global correlation, it achieves relatively lower regret than other signals, allowing us to incorporate it with a minimal weight to boost selection performance without distorting the global value estimation. In comparison, the rollout consistency and confidence signals underperform on both global and local metrics. Ultimately, the fused signal $Q(s)$ attains the optimal equilibrium, maintaining high global alignment while simultaneously achieving low local regret, thereby justifying the adoption of the fusion mechanism in Section 3.2.

4.4 Computational Efficiency Analysis

To evaluate asynchronous acceleration, efficiency tests were conducted on a 48GB NVIDIA RTX 5880 Ada GPU with fixed random seeds. We benchmark against **ToT** (BFS-based beam search), **rStar** (generator only), and **Sync-MCTS** (a synchronous variant of our method). To ensure fair comparison, we standardized the search budget by capping LLM calls at approximately 180 for all methods.

Latency Reduction Table 3 demonstrates the superior efficiency gains of our proposed approach. By parallelizing computationally intensive self-evaluation tasks including process and pairwise rewards, ASE-MCTS achieves a substantial $2.8\times$ to $4.5\times$ speedup relative to the synchronous baseline Sync-MCTS. Furthermore, despite incorporating complex reward signals, ASE-MCTS remains significantly faster than both ToT and rStar. This

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 3: Computation time in seconds across models and methods. ASE-MCTS achieves the best performance across all models and tasks, significantly outperforming the baselines.

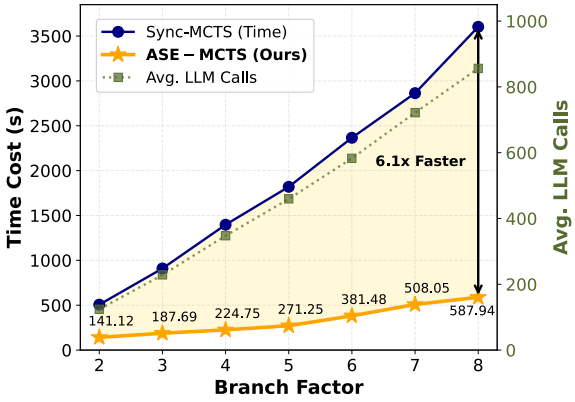


Figure 6: Latency comparison between Sync-MCTS and ASE-MCTS under varying branch factors on AMC2023. While the computational cost (Avg. LLM Calls) increases linearly, ASE-MCTS significantly suppresses latency growth, achieving a $6.1\times$ speedup at a branch factor of 8.

confirms that our asynchronous architecture successfully masks the evaluation overhead, preventing the "reasoning pause" typically observed in synchronous search algorithms while maintaining high-quality exploration.

Scalability with Computational Load While batch inference (via vLLM (Kwon et al., 2023)) renders the latency of rollout sampling negligible, the **branch factor** remains the primary computational bottleneck, as it geometrically expands the search space and the volume of concurrent self-evaluations. We analyzed the average processing time per case on AMC2023 using Qwen2.5-7B-Instruct. As illustrated in Figure 6, the latency of the synchronous baseline exhibits a linear growth trajectory that directly correlates with the increasing volume of LLM calls. In contrast, ASE-MCTS exhibits a remarkably flat latency curve by effectively hiding these costs via asynchronous parallelization. Notably, at a branch factor of 8, our method achieves a speedup of up to $6.1\times$, demonstrating exceptional scalability.

5 Conclusion

In this paper, we presented ASE-MCTS, a fully self-evaluated MCTS framework without any external supervision. By incorporating a Prompt-Guided Action Space and a Hierarchical Diverse Expansion strategy, ASE-MCTS ensures a comprehensive exploration of the solution space. Crucially, through the fusion of three distinct self-evaluation signals, we successfully balance global alignment and local discriminability of reward signal. Furthermore, we introduced an asynchronous orchestrator to maximize task parallelism, significantly reducing the latency associated with expansion and simulation.

Comprehensive evaluations across rigorous reasoning benchmarks including GPQA-Diamond and Olympiad-Physics reveal that ASE-MCTS significantly surpasses competitive baselines such as Self-Consistency and rStar. Notably, the proposed framework yields up to a $6\times$ speedup relative to conventional synchronous architectures. These findings confirm the immense reasoning potential of self-evaluated MCTS and offer a practical path toward deploying autonomous, high-performance reasoning agents in resource-constrained environments.

Limitations

While ASE-MCTS significantly improves reasoning efficiency and accuracy, it still has some limitations.

First, the rigid decomposition of reasoning steps poses a trade-off between search structure and cognitive coherence. Artificially segmenting the thought process can disrupt the model's natural flow, potentially degrading performance on tasks requiring holistic intuition.

Second, although our asynchronous design effectively masks latency, it does not mitigate the substantial aggregate token consumption and computational load. The extensive sampling required by the search tree creates severe pressure on GPU memory. Future research could explore advanced memory

optimization techniques, such as KV cache sharing, to make tree search more cost-effective and deployable.

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, and 1 others. 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, Zengmao 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.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Amy Yang, Angela Fan, and 1 others. 2024. The llama 3 herd of models. *arXiv 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, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

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

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, and 1 others. 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*.

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

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. 2023. 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 Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, and 1 others. 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*.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, and 25 others. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

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, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

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

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, and 1 others. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623.

A Experiment Supplements

To ensure statistical reliability, we report the mean performance across multiple runs whenever computational resources permitted. However, due to the prohibitive time costs associated with certain baselines (e.g., rStar), the number of trials for these methods was restricted. Notably, our reproduced results may exhibit deviations—specifically higher performance scores—compared to prior literature. This discrepancy is primarily attributed to our evaluation methodology; unlike rigid rule-based matching, our verification protocol effectively identifies correct answers that are frequently misclassified as false negatives by standard scripts. Further details regarding this evaluation mechanism are provided in Appendix A.2.

A.1 Dataset Selection

To rigorously evaluate the reasoning capabilities enhanced by our MCTS framework, we prioritized datasets that offer high complexity and varying degrees of difficulty. While datasets such as GSM8K and ProntoQA are widely utilized in prior MCTS research, our preliminary experiments indicated that the three state-of-the-art Small Language Models (SLMs) employed in this study already exhibit near-saturation performance on these benchmarks. Consequently, these datasets fail to provide sufficient headroom to observe the incremental benefits of inference-time search strategies. Furthermore, regarding StrategyQA (utilized in works like rStar),

our empirical analysis revealed a lack of significant discriminative power across various models and baselines in our specific experimental setup.

Therefore, we selected a suite of challenging benchmarks designed to test multi-step logical deduction and domain-specific knowledge across three distinct dimensions:

- **Mathematical Reasoning:** We employ MATH500, AMC2023, and AIME2024. These datasets represent a gradation of difficulty from challenging high school problems to olympiad-level competitions, requiring complex reasoning chains that are ideal for MCTS evaluation.

- **Scientific Reasoning:** we integrate GPQA-Diamond and Olympiad-Physics. GPQA-Diamond provides graduate-level, domain-resistant questions that probe intrinsic knowledge and multi-step logical inference beyond simple retrieval. Olympiad-Physics further stresses the search process by requiring the correct application of physical laws under intricate constraints, combining domain knowledge with deep derivational reasoning.

A.2 Answer Verification

Standard answer matching metrics, such as Exact Match, are often overly rigid for complex reasoning tasks. Correct answers generated by models are frequently penalized due to superficial formatting differences (e.g., LaTeX syntax, units, decimal vs. fraction representation, or ordering), leading to an underestimation of the model's true performance. To address this and ensure a more accurate judgment of correctness, we implemented a composite verification pipeline consisting of heuristic rules, LLM-based judging, and human review:

- **Heuristic Rule Filtering:** Before model-based verification, we apply deterministic rules to handle numeric and boolean values. We set a tolerance threshold of 10^{-5} for floating-point comparisons and align boolean logic to filter out obvious errors and maintain numeric rigor.

- **Dual-Model Semantic Equivalence Check:** For answers that fail string matching but are not rejected by heuristics, we employ two distinct Large Language Models, **DeepSeek-v3.2** and **Qwen3-Max**, as semantic judges. We explicitly instruct these models to ignore formatting nuances (such as whitespace, capitalization, or unit styles) and focus solely on whether the extracted answer is semantically equivalent to the ground truth.

- **Human Verification:** To mitigate potential hallucinations or overly lenient judgments (false positives) by the LLMs, we incorporate a final human verification step. All samples flagged as "Correct" by the LLMs are manually reviewed to ensure the fairness and reliability of the final results.

A.3 Details of Reward

Global Pearson Correlation Analysis. Figure 7 illustrates the global Pearson correlation matrix across diverse evaluation metrics. **Notably, the sub-matrix in the bottom-right quadrant is identical to the visualization presented in Figure 5**, highlighting the substantial consensus among advanced Large Language Models (LLMs), including DeepSeek, GPT, Qwen, and DouBao. In this analysis, we establish **ET**—an ensemble metric derived from these four robust models—as the primary **reference standard** for **assessing the quality of evaluation metrics**.

Performance of Individual Signals. Among the proposed internal signals, `rollout_reward` and `process_score` demonstrate **strong alignment** with the ET reference, yielding correlation coefficients of 0.52 and 0.46, respectively. These findings suggest that both the validity of the final solution and the quality of the reasoning process are critical indicators for assessing the quality of reasoning paths.

Efficacy of Signal Fusion. The **superior performance of** `fusion_score` validates the effectiveness of a multi-dimensional evaluation strategy. By integrating `process_score`, `rollout_reward`, and `pairwise_score` via a weighted ensemble mechanism, this composite metric achieves an enhanced correlation of 0.58 with ET, surpassing the performance of the most effective individual metric (`rollout_reward`). This improvement underscores a synergistic effect: the fusion strategy leverages **the complementarity of multiple perspectives** to effectively mitigate the inherent noise and variance of single indicators. Consequently, it constructs a more stable and reliable evaluation proxy

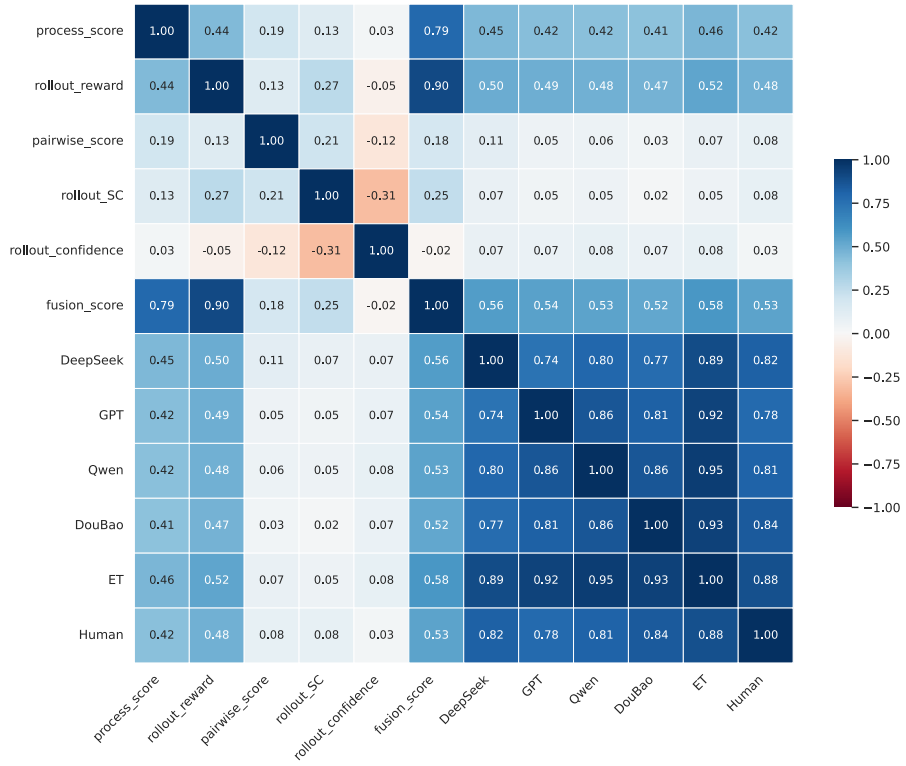


Figure 7: Global Pearson correlation analysis. This heatmap illustrates the macro-level consistency across different evaluation metrics and datasets.

that maintains a higher degree of consistency with the collective judgment of strong LLMs.

B Implementation Details

B.1 Efficiency Strategies

Given the vast action space and the difficulty of determining termination in open-ended reasoning, minimizing computational waste is paramount. We implement two optimization mechanisms. First, we introduce an Inline Terminal Check. Instead of invoking a separate model call to judge completion, we instruct the LLM to output a specific token sequence (e.g., "Check Terminal: Yes") within the generation if the problem is resolved. This allows us to instantly identify leaf nodes without additional overhead. Second, we implement Dynamic Rollout Allocation. As the search depth increases and the reasoning trajectory becomes more complete, the marginal information gain from expensive lookahead simulations diminishes. Consequently, we progressively reduce the number of rollout simulations for deeper nodes, allocating computational budget where it is most impactful.

B.2 Hyperparameter Settings

In this section, we detail the key hyperparameters used in ASE-MCTS. The configuration is categorized into search parameters, reward fusion weights, and answer aggregation weights.

Search Configuration These parameters control the structure and depth of the tree search:

- `iterations` (Default: 15): Specifies the total number of MCTS simulation rounds. We typically set this between 15 and 20 to balance performance and latency.
- `exploration_constant` (Default: 12): Corresponds to the c_{puct} term in the UCT algorithm. Since our reward signals are normalized to the range $[0, 10]$ (as opposed to the standard $[0, 1]$), we scale the exploration constant by approximately $10\times$ to ensure adequate exploration of unvisited nodes.
- `branch_factor_init` (Default: 4) & `branch_factor` (Default: 3): These control the breadth of the search tree. `branch_factor_init` determines the number of child nodes expanded from the root using high-temperature repeated sampling.

`branch_factor` controls the expansion of subsequent nodes using our *Hierarchical Diverse Expansion* strategy. For tasks with higher difficulty, increasing these factors allows for broader coverage of the solution space.

- `rollout_num` (Default: 3): The number of lookahead simulation paths generated during the evaluation of each leaf node. This directly impacts the reliability of the rollout reward.

- `max_depth` (Default: 10): The maximum allowable depth for the search tree, preventing infinite loops in reasoning chains.

Reward Signal Fusion The following weights determine the contribution of each signal to the final node value $Q(s)$.

- `rollout_weight` (0.5) & `process_weight` (0.4): These serve as the primary global anchors for value estimation. We generally assign a higher weight to the rollout reward, particularly when `rollout_num` is sufficient to provide a stable estimate.

- `pairwise_weight` (0.1): Used for fine-grained local discrimination. A smaller weight is sufficient to break ties between high-quality candidates without distorting the global value.

- `rollout_sc_weight` (0.01) & `rollout_confidence_weight` (0.01): Auxiliary signals based on semantic consistency and token probabilities. In our asynchronous orchestrator, if a weight is set to 0, the corresponding evaluation task is never created, thereby saving computational resources. These are typically set to negligible values or zero in standard runs.

Answer Aggregation After the search concludes, the final answer is selected based on a weighted combination of metrics:

- `leaf_confidence_weight` (0.4): Functions similarly to a majority vote mechanism. While reliable for simpler problems, it is less effective for complex tasks where correct solutions are sparse.

- `leaf_value_weight` (0.3) & `path_value_weight` (0.3): These weights prioritize the intrinsic quality of the reasoning chain (accumulated node values) and the final leaf node value, ensuring that the selected answer is not just popular, but rigorously reasoned.

C Prompt Templates

We implement all prompts using a `chat_template` that explicitly delineates `<|system|>`, `<|user|>`, and `<|assistant|>` message segments, instead of writing a single free-form prompt in a direct LLM API call. The key difference is that `chat_template` enforces a clear separation between global behavioral constraints (`system_prompt`) and instance-specific task inputs (`user_prompt`), while supporting an explicit context carrier (`prefix`) for previously generated steps; this separation reduces ambiguity and improves format stability for downstream parsing. Each template is parameterized by four fields: `prefix`, `system_prompt`, `user_prompt`, and `suffix`. Crucially, the `suffix` is placed at the *beginning* of the assistant's output so that the model is forced to *start* its answer with the prescribed headers (e.g., `<step>` or `<ideas>` and fixed key fields), thereby enhancing instruction following and making the output easier to parse reliably.

C.1 ASE-MCTS prompts

We use eight prompt templates to support hierarchical expansion, rollout generation, and multi-view self-evaluation. Figures 8–15 show the exact prompt texts.

- **Initial-Expand Prompt** (Figure 8). This prompt initializes reasoning by generating *only the first step* for a given question. It enforces a strict `<step>` container with a concise `[Core idea]` followed by the full content of that step, ensuring a uniform step representation from the start.

- **Diverse-Ideas Prompt** (Figure 9). This prompt generates multiple mutually distinct core ideas for the subsequent reasoning step conditioned on the current partial trajectory. By enforcing diversity at the step-intent level before expansion, it reduces redundancy among candidate continuations and increases the diversity of explored branches during deeper tree search.

- **Idea-Expand Prompt** (Figure 10). Conditioned on the existing steps and one selected direction, this prompt expands *one* core idea into a complete next step. It also performs an inline termination check by outputting a dedicated `[Check terminal]` field to indicate whether the node is terminal.

- **Fast-Rollout Prompt** (Figure 11). This prompt completes the remaining reasoning from the current partial path *to the end* in a single continuation. It is primarily used for efficient rollout to obtain candidate full solutions for outcome-based scoring.

- **Process-Evaluation Prompt** (Figure 12). This prompt evaluates the *current step* against predefined process criteria and returns a numeric score (0–10) with a brief justification. It operationalizes the process reward by measuring local correctness, coherence, and compliance at the step level.

- **Reward-Evaluation Prompt** (Figure 13). This prompt evaluates the *full solution* against task objectives and returns a numeric score (0–10) with justification. It serves as an outcome-oriented signal to estimate global solution quality for trajectories produced by rollouts.

- **Pair-Evaluation Prompt** (Figure 14). Given two candidate steps under the same context, this prompt performs a *pairwise comparison* and decides which step is better (A/B/Tied) with a margin-like score (0–5). It is designed to improve local discriminability among sibling nodes during selection.

- **Best-Answer Prompt** (Figure 15). This prompt transforms the highest-scoring solution into the final output and enforces a strict `\boxed{...}` format. It is applied at the aggregation stage to generate the submission-ready answer corresponding to the selected trajectory.

C.2 Search Guide

The search guide is a dataset-conditioned configuration that instantiates the placeholders in our prompt templates and thereby steers the MCTS search behavior. Concretely, it specifies step-generation hints, expansion constraints, and evaluation rubrics that are injected into the corresponding prompts, so that the same search procedure can adapt to different domains and answer formats.

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."*

D Use of AI Assistants

We acknowledge the use of Large Language Models in this work. Specifically, DeepSeek-V3.2 was used to assist in writing and debugging parts of the implementation code. Furthermore, Gemini 3 Pro was utilized to polish the writing and improve the clarity of the text. All scientific claims and experimental results remain the sole responsibility of the authors.

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.

Diverse Ideas 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: Diverse-Ideas 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 criterions (**[pairwise criterions]**).

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 criterions 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.