DeepSearch: Overcome the Bottleneck of Reinforcement Learning with Verifiable Rewards via Monte Carlo Tree Search

Fang Wu$^{♡*}$ **Weihao Xuan**$^{▽,△*}$ **Heli Qi**$^{△*}$ **Ximing Lu**$^{♢}$ **Aaron Tu**$^{♠}$
Li Erran Li$^{♣}$ **Yejin Choi**$^{♡†}$
$^{♡}$Stanford University $^{▽}$University of Tokyo $^{△}$RIKEN AIP $^{♢}$University of Washington
$^{♠}$UC Berkeley $^{♣}$Amazon AWS

Abstract

Although Reinforcement Learning with Verifiable Rewards (RLVR) has become an essential component for developing advanced reasoning skills in language models, contemporary studies have documented training plateaus that emerge following thousands of optimization steps, demonstrating notable decreases in performance gains despite increased computational investment. This limitation stems from the sparse exploration patterns inherent in current RLVR practices, where models rely on limited rollouts that often miss critical reasoning paths and fail to provide systematic coverage of the solution space. We present DeepSearch, a framework that integrates Monte Carlo Tree Search (MCTS) directly into RLVR training. In contrast to existing methods that rely on tree search only at inference, DeepSearch embeds structured search into the training loop, enabling systematic exploration and fine-grained credit assignment across reasoning steps. Through training-time exploration, DeepSearch addresses the fundamental bottleneck of insufficient exploration, which leads to diminishing performance improvements over prolonged training steps. Our contributions include: (1) a global frontier selection strategy that prioritizes promising nodes across the search tree, (2) selection with entropy-based guidance that identifies confident paths for supervision, and (3) adaptive replay buffer training with solution caching for efficiency. Experiments on mathematical reasoning benchmarks show that DeepSearch achieves 62.95% average accuracy and establishes a new state-of-the-art for 1.5B reasoning models - a 1.25 percentage point improvement over the previous best while using 5.7x fewer GPU hours than extended training approaches. These results highlight the importance of strategic exploration over brute-force scaling and demonstrate the promise of algorithmic innovation for advancing RLVR methodologies. DeepSearch establishes a new direction for scaling reasoning capabilities through systematic search rather than prolonged computation.

🤗 https://huggingface.co/fangwu97/DeepSearch-1.5B

1 Introduction

Large language models (LLMs) have recently achieved notable progress on complex reasoning tasks (DeepSeek-AI, 2025; Yang et al., 2024), driven in part by test-time computation scaling strategies (Li et al., 2023; Yao et al., 2023; Bi et al., 2024; Zhang et al., 2024a; Guan et al., 2025) such as tree search with process-level evaluation. While effective, these methods typically treat structured search as an inference-only mechanism, leaving untapped potential to integrate systematic exploration into the training process itself.

This separation between training and inference creates fundamental limitations in how we scale reinforcement learning with verifiable rewards (RLVR) for reasoning. Current RLVR approaches remain

---

*Equal contributions.

†Corresponding author. Email: yejinc@stanford.edu

constrained by sparse exploration patterns during training (Wu et al., 2025; Liu et al., 2025c), while models are expected to demonstrate sophisticated search behaviors only at inference time. Even recent advances in prolonged RL training (Liu et al., 2025a) have shown performance plateaus after thousands of steps, with clear diminishing returns from allocating more computation to additional training depth. This suggests that simply scaling the number of training steps—the primary axis explored in prior work—may not be sufficient to unlock the full potential of RLVR.

We address this gap by introducing DeepSearch, a framework that embeds Monte Carlo Tree Search (MCTS) (Metropolis & Ulam, 1949) directly into RLVR training, representing a fundamental shift from scaling training depth to scaling training breadth. By coupling structured search with verifiable rewards during training, DeepSearch enables models to learn not only from correct solutions but also from the systematic exploration process itself, providing richer supervision than outcome-based or direct rollout methods (Lyu et al., 2025; He et al., 2025b).

The core insight driving us is to *focus on training-time exploration* as the driver of improved reasoning. While traditional RLVR relies on limited rollouts that may miss critical reasoning paths, DeepSearch systematically expands the reasoning frontier during training through principled tree search. This design advances three key objectives: *(i)* expanding reasoning coverage beyond what direct policy rollouts can achieve, *(ii)* providing fine-grained credit assignment to intermediate reasoning steps through tree-structured backpropagation, and *(iii)* maintaining computational efficiency through intelligent node selection and solution caching strategies.

To achieve these goals, DeepSearch introduces several key innovations. First, *global frontier selection* strategy prioritizes the most promising nodes across the entire search tree, moving beyond traditional root-to-leaf UCT traversals that can be computationally wasteful and myopic. Second, *selection with entropy-based guidance* systematically identifies confident incorrect reasoning paths for supervision. Finally, an adaptive training strategy with replay buffers progressively filters challenging problems and caches verified solutions to avoid redundant computation across training iterations.

We evaluate DeepSearch on mathematical reasoning benchmarks, where it significantly outperforms state-of-the-art RLVR baselines, including Nemotron-Research-Reasoning-Qwen-1.5B v2 (Liu et al., 2025a) and DeepScaleR (Luo et al., 2025b). Our results show that DeepSearch achieves 62.95% average accuracy on challenging mathematical tasks, representing **a new state-of-the-art for 1.5B reasoning models**. Importantly, these gains are achieved while remaining computationally efficient through progressive filtering and intelligent solution reuse, demonstrating that search-augmented training can be both more effective and more practical than conventional approaches.

The implications extend beyond math reasoning: by bridging the gap between inference-time search capabilities and training-time learning, DeepSearch establishes a new paradigm for scaling RLVR that emphasizes systematic exploration over prolonged training. This work suggests that the future of reasoning model development lies not just in scaling model parameters or training steps, but in fundamentally rethinking how we structure the learning process to mirror the sophisticated reasoning patterns we expect at inference time. We defer a detailed literature review to Appendix A due to space constraints.

2 DeepSearch with MCTS

Given a problem x and a policy model π_θ, we adopt a modified MCTS framework to build a search tree for incremental step-by-step solution exploration. We replace traditional root-to-leaf selection with global frontier-based node selection. The root node represents the question x, and child nodes correspond to intermediate steps s generated by π_θ. A root-to-leaf path ending at a terminal node s_{end} forms a trajectory $\mathbf{t} = x \oplus s_1 \oplus s_2 \oplus \ldots \oplus s_{\text{end}}$, where each step s_i is assigned a q-value $q(s_i)$. Then we extract solution trajectories $\mathbb{T} = \{\mathbf{t}^1, \mathbf{t}^2, \ldots, \mathbf{t}^n\}$ $(n \geq 1)$ from the search tree $\mathcal{T}$, where $\mathbf{t}^i$ can be correct, incorrect or incomplete. The depth of any node s is denoted as $d(s) \in \mathbb{Z}^+$. $N(s)$ and $\xi(s)$ denote the number of visits to s and the number of children nodes of s, respectively. Starting from the root node x, our MCTS iterations are conducted through four subsequent components.

2.1 Expansion with Entropy-based Guidance

In step i, we collect the latest reasoning trajectory $o_i = x \oplus s_1 \oplus s_2 \oplus \ldots \oplus s_{i-1}$ as the current state, i.e., observation. Based on this state, we prompt the policy model $\pi_\theta(s_i|o_i)$ to generate n candidates
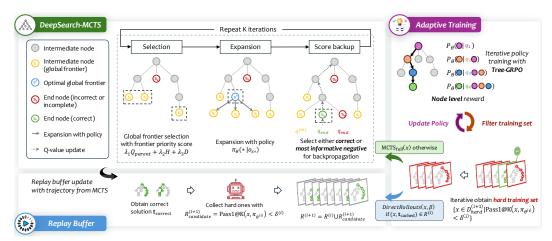
Figure 1: DeepSearch Framework Overview.

for the next-step reasoning trail $\{s_{i,j}\}_{j=1}^n$. We repeat this expansion behavior until we reach the terminal nodes $s_{\text{end}} \in \mathcal{S}_{\text{end}}$, either by arriving at the final answers or by hitting the maximum depth of the tree $d_{\mathcal{T}}$, which yields an ordered sequence $s_1 \to \cdots \to s_{\text{end}}$.

During each expansion, let $\mathcal{S}_{\text{end}}^{(k)}$ denote the set of newly generated terminal nodes at iteration k. We evaluate the correctness of each terminal node using a verification function $\mathcal{V} : \mathcal{S}_{\text{end}} \to \{0, 1\}$, where $\mathcal{V}(s) = 1$ indicates a correct solution and $\mathcal{V}(s) = 0$ indicates an incorrect or incomplete solution. Then we partition the terminal nodes into correct and incorrect subsets:

$$\mathcal{S}_{\text{correct}}^{(k)} = \{s \in \mathcal{S}_{\text{end}}^{(k)} \mid \mathcal{V}(s) = 1\}, \quad \mathcal{S}_{\text{incorrect}}^{(k)} = \{s \in \mathcal{S}_{\text{end}}^{(k)} \mid \mathcal{V}(s) = 0\}. \tag{1}$$

If $\mathcal{S}_{\text{correct}}^{(k)} = \emptyset$, we employ an *entropy-based selection* to identify the most confident negative example, where the terminal node with the lowest average entropy along its root-to-leaf trajectory is selected:

$$s_{\text{neg}}^* = \arg \min_{s \in \mathcal{S}_{\text{incorrect}}^{(k)}} \bar{H}(\mathbf{t}(s)), \tag{2}$$

where $\mathbf{t}(s) = (x, s_1, s_2, \ldots, s)$ represents the unique trajectory from root x to terminal node s, and the average trajectory entropy is defined as:

$$\bar{H}(\mathbf{t}(s)) = \frac{1}{|\mathbf{t}(s)|} \sum_{i=1}^{|\mathbf{t}(s)|} H(\pi_\theta(s_i \mid o_i)), \tag{3}$$

with final $H(\pi_\theta(s_i \mid o_i)) = -\sum_{a_{i,k}} \pi_\theta(a_{i,k} \mid o_i, a_{i,<k}) \log \pi_\theta(a_{i,k} \mid o_i, a_{i,<k})$ being the Monte Carlo estimation of the Shannon entropy of the policy distribution at step i. $a_{i,k}$ is the k-th token of step s_i, and $a_{i,<k}$ denotes the tokens preceding $a_{i,k}$. This selection strategy prioritizes incorrect reasoning sequences exhibiting low decision uncertainty, targeting areas where the model's decision-making is most confident and would benefit from additional training supervision.

2.2 HEURISTIC SCORE BACKUP

Let $\mathbf{t}^*$ denote the selected trajectory for backpropagation, which is either a correct solution trajectory or the most confident negative trajectory $\mathbf{t}(s_{\text{neg}}^*)$ identified through entropy-based selection. Let $q^{(m)}(s_i)$ denote the *q-value* for node $s_i \in \mathbf{t}^*$ after the m-th rollout backpropagation. We define the iterative q-value update rule for nodes along the selected trajectory:

$$q^{(m)}(s_i) = q^{(m-1)}(s_i) + \gamma(i, l) \cdot q^{(m)}(s_{\text{end}}), \tag{4}$$

where $\gamma(i, l) : \mathbb{Z}^+ \times \mathbb{Z}^+ \to [0, 1]$ is the *temporal decay* function that assigns higher weights to nodes closer to the terminal node:

$$\gamma(i, l) = \max\left(\frac{i}{l}, \gamma_{\min}\right), \tag{5}$$

with i being the current node index in the trajectory, l being the terminal node index, and $\gamma_{\min} = 0.1$ being the minimum decay threshold.

The q-value initialization is $q^{(0)}(s_i) = 0$ for all $s_i \in \mathcal{T}$. Terminal node rewards are assigned according to the verification function's result:

$$q(s_{\text{end}}) = \begin{cases} +1 & \text{if } \mathcal{V}(s_{\text{end}}) = 1 \text{ (correct)}, \\ -1 & \text{if } \mathcal{V}(s_{\text{end}}) = 0 \text{ (incorrect)} \vee d(s_{\text{end}}) < d_{\mathcal{T}} \text{ (incomplete)}. \end{cases} \tag{6}$$

To ensure positive q-values (*e.g.*, $q_{\text{correct}} = 0.1$) for nodes on correct reasoning paths while penalizing nodes leading to incorrect or incomplete solutions, we enforce the constrained update rule:

$$q^{(m)}(s_i) = \begin{cases} q^{(m-1)}(s_i) + \gamma(i,l) \cdot q^{(m)}(s_{\text{end}}) & \text{if } q^{(m-1)}(s_i) \cdot q^{(m)}(s_{\text{end}}) \geq 0, \\ \gamma(i,l) \cdot q^{(m)}(s_{\text{end}}) & \text{elif } q^{(m)}(s_{\text{end}}) > 0, \\ q^{(m-1)}(s_i) & \text{elif } q^{(m-1)}(s_i) > 0. \end{cases} \tag{7}$$

This constraint preserves the invariant that $q^{(m)}(s_i) \geq 0$ for all intermediate nodes $s_i \in \mathcal{T} \setminus \mathcal{S}_{\text{end}}$ lying on trajectories leading to correct solutions, while allowing negative values only for nodes that inevitably lead to incorrect outcomes.

2.3 HYBRID SELECTION STRATEGY

Our MCTS employs a *hybrid selection strategy* that combines traditional UCT-based local selection with novel global frontier selection, each serving distinct purposes in the search process.

Local Selection for Sibling Comparison During the expansion of a selected node, we generate multiple candidate children and need to determine which ones to add to the tree. For this *local sibling comparison*, we follow the traditional MCTS protocol and employ the Upper Confidence Bounds for Trees (UCT) algorithm (Kocsis & Szepesvári, 2006):

$$\text{UCT}(s) = Q(s) + \lambda \sqrt{\frac{\ln N_{\text{parent}}(s)}{N(s)}}, \tag{8}$$

where $Q(s) = \frac{q(s)}{N(s)}$ represents the average reward per visit, $N_{\text{parent}}(s)$ is the number of visits from the parent node, and λ balances exploitation and exploration. This local selection ensures that we make optimal decisions when choosing among sibling nodes that share the same parent and context.

Global Frontier Selection for Next Expansion After completing the first score backup phase, we need to identify the most promising node across the *entire search tree* for the next expansion round. This is where our novel global frontier selection mechanism operates.

Unlike traditional MCTS, which performs root-to-leaf traversals using UCT at each level, our global approach directly compares all frontier nodes simultaneously. We maintain a global view of all leaf nodes across the entire search tree $\mathcal{T}$ and prioritize promising expansion points globally:

$$\mathcal{F} = \{s \in \mathcal{T} \mid \xi(s) = 0, s \notin \mathcal{S}_{\text{end}}, d(s) < d_{\mathcal{T}}\}. \tag{9}$$

For each frontier node $s \in \mathcal{F}$, we compute a *frontier priority score*:

$$F(s) = \underbrace{\lambda_1 \times \tanh(Q_{\text{parent}}(s))}_{\text{Quality Potential}} + \underbrace{\lambda_2 \times H(\pi_\theta(s|o))}_{\text{Uncertainty Bonus}} + \underbrace{\lambda_3 \times D(d(s))}_{\text{Depth Bonus}}. \tag{10}$$

Here, the quality potential term $\tanh(Q_{\text{parent}}(s))$ encourages the selection of nodes whose parents have demonstrated high value, using the tanh transformation to smoothly handle negative Q-values and map them to the range $[-1, 1]$. The uncertainty bonus term $H(\pi_\theta(s|o))$ provides exploration guidance by adjusting priority according to the policy's entropy; the sign of its coefficient can be utilized to steer selection toward regions with high confidence or uncertainty. The depth bonus term $D(d(s))$ encourages deeper exploration by providing additional priority to nodes at greater depths, where we empirically find $D(d(s)) = \sqrt{d(s)/d_{\mathcal{T}}}$ to be most effective among other variants including $d(s)$ and $\log(d(s) + 1)$. The node with the highest frontier score is selected for the next expansion: $s^* = \arg\max_{s \in \mathcal{F}} F(s)$.

4

Rationale for Hybrid Approach This hybrid design leverages complementary strengths: local UCT selection ensures principled sibling comparisons within subtrees, while global frontier selection overcomes UCT's myopia through cross-subtree resource allocation. The approach achieves three key advantages: *(1) Computational efficiency* by eliminating redundant root-to-leaf traversals, *(2) Enhanced exploration coverage* by preventing the algorithm from getting trapped in locally promising but globally suboptimal subtrees, and *(3) Uncertainty-guided search* that leverages the policy's entropy to target regions expected to benefit from additional training supervision, with the bonus coefficient controlling the direction of this preference.

3 ADAPTIVE TRAINING STRATEGY WITH REPLAY BUFFER

While MCTS offers fine-grained credit assignment, applying it to every training example is computationally infeasible. To address this, we adopt an iterative filtering strategy with a replay buffer mechanism that focuses MCTS computation on challenging examples while preventing catastrophic forgetting of solved problems. The complete pipeline is depicted in Algorithm 1.

3.1 ITERATIVE TRAINING WITH PROGRESSIVE FILTERING

Our training process follows an iterative approach that progressively refines the training subset based on model performance. We begin by using the base RL model to perform an initial screening on the entire dataset $\mathcal{D}_{\text{hard}}$, creating the first training subset $\mathcal{D}_{\text{hard}}^{(0)}$ for MCTS-based RL training.

Specifically, the iterative training process proceeds as follows:

Initial Subset Construction: Given the base policy $\pi_{\theta^{(0)}}$, we evaluate its performance on the full training set $\mathcal{D}_{\text{train}}$ using direct rollouts and construct the initial hard subset:

$$\mathcal{D}_{\text{hard}}^{(0)} = \{x \in \mathcal{D}_{\text{train}} \mid \texttt{Pass1@K}(x, \pi_{\theta^{(0)}}) < \delta^{(0)}\}, \tag{11}$$

where $\texttt{Pass1@K}(x, \pi)$ represents the success rate when sampling $K = 4$ solutions for problem x using policy π, and $\delta^{(0)} \in (0, 1)$ is the initial filtering threshold.

Iterative Refinement: After each training phase i, we re-evaluate the updated policy $\pi_{\theta^{(i)}}$ on the current hard subset and apply threshold-based filtering to create the next iteration's training set:

$$\mathcal{D}_{\text{hard}}^{(i+1)} = \{x \in \mathcal{D}_{\text{hard}}^{(i)} \mid \texttt{Pass1@K}(x, \pi_{\theta^{(i)}}) < \delta^{(i)}\}. \tag{12}$$

The filtering threshold $\delta^{(i)}$ is typically set to 25%, ensuring that only problems with insufficient success rates remain in the active training set. This progressive filtering concentrates computational resources on increasingly challenging problems as the model improves.

3.2 REPLAY BUFFER WITH CACHED SOLUTIONS

To prevent catastrophic forgetting and efficiently leverage previously discovered solutions, we maintain a replay buffer $\mathcal{R}$ that stores correct reasoning trajectories from earlier training phases.

Buffer Population. During each training iteration i, we identify problems that obtained correct solutions through MCTS rollouts but still fail to meet the filtering threshold after training:

$$\mathcal{R}_{\text{candidates}}^{(i)} = \{(x, \mathbf{t}_{\text{correct}}) \mid x \in \mathcal{D}_{\text{hard}}^{(i)}, \exists \mathbf{t}_{\text{correct}} \in \mathbb{T}(x), \texttt{Pass1@K}(x, \pi_{\theta^{(i)}}) < \delta^{(i)}\}. \tag{13}$$

These candidate trajectories are added to the replay buffer, attaining $\mathcal{R}^{(i+1)} = \mathcal{R}^{(i)} \cup \mathcal{R}_{\text{candidates}}^{(i)}$.

Cached Solution Usage. Instead of randomly sampling from the replay buffer, we employ a deterministic strategy that directly utilizes cached solutions when available. For each problem x in the current training iteration, we first check whether a correct solution has been previously cached. This approach eliminates redundant MCTS computation for problems with known solutions while focusing computational resources on truly challenging unsolved problems.

Hybrid Rollout Strategy. When processing problems in the current hard subset $\mathcal{D}_{\text{hard}}^{(i)}$, we apply different rollout strategies based on cache availability:

$$\text{Rollout}(x) = \begin{cases} \mathbf{t}_{\text{cached}} \cup \text{DirectRollouts}(x, \beta) & \text{if } (x, \mathbf{t}_{\text{cached}}) \in \mathcal{R}^{(i)}, \\ \text{MCTS}_{\text{full}}(x) & \text{otherwise.} \end{cases} \tag{14}$$

For problems with cached solutions, we directly incorporate the stored correct trajectory $\mathbf{t}_{\text{cached}}$ and supplement it with $\text{DirectRollouts}(x, \beta)$, which samples $\beta \cdot B$ additional solution attempts from the current policy $\pi_\theta(\cdot|x)$, where $0 < \beta < 1$ and B is the standard sampling budget. For problems without cached solutions, we apply the complete MCTS search process $\text{MCTS}_{\text{full}}(x)$. Moreover, among the incorrect samples, we remove data containing garbled text or infinite repetitions. Based on empirical evidence, optimizing policies on such problematic data frequently leads to training collapse (Bai et al., 2025). The training dataset for each iteration is then constructed as:

$$
\mathcal{T}_{\text{train}}^{(i)} = \underbrace{\bigcup_{x:(x, \mathbf{t}_{\text{cached}}) \in \mathcal{R}^{(i)}} \{\mathbf{t}_{\text{cached}} \cup \text{DirectRollouts}(x, \beta)\}}_{\text{Cached problems}} \cup \underbrace{\bigcup_{x:(x, \mathbf{t}_{\text{cached}}) \notin \mathcal{R}^{(i)}} \text{MCTS}_{\text{full}}(x)}_{\text{Unsolved problems}}.
\tag{15}
$$

This construction eliminates the need for artificial sampling ratios or complex batch composition strategies, as training data naturally incorporate both preserved knowledge and fresh exploration based on problem-specific requirements. This achieves three key benefits: *(1) Computational efficiency* by avoiding redundant MCTS computation, *(2) Solution preservation* by guaranteeing the inclusion of cached correct trajectories, and *(3) Continued exploration* at minimal computational cost.

3.3 TREE-GRPO TRAINING OBJECTIVE

After constructing a search tree $\mathcal{T}$ for a sample question x in the dataset $\mathcal{D}_{\text{train}}$, we develop our Tree-GRPO training objective. This objective combines q-value regularization with policy optimization to effectively learn from the tree-structured reasoning traces.

Q-Value Soft Clipping. To address the q-value explosion problem for intermediate nodes while preserving meaningful gradients, we first apply *soft clipping* using the hyperbolic tangent function:

$$
q(s_j) = \tanh\left(q^{(k_{\max})}(s_j)/\epsilon_q\right) \cdot q_{\max} \quad \text{for all } s_j \in \mathcal{T} \setminus \mathcal{S}_{\text{end}}
\tag{16}
$$

where $k_{\max}$ is the maximum rollout iterations, $\epsilon_q = 1.0$ is the temperature parameter, and $q_{\max} = 1$ defines the maximum allowable q-value magnitude.

This soft clipping approach prevents q-value explosion by maintaining all intermediate node q-values within $[-q_{\max}, q_{\max}]$, while offering several key advantages: *(i)* it naturally bounds q-values without hard discontinuities, *(ii)* it preserves gradients everywhere, preventing the zero-gradient problem that occurs with hard clipping when all values hit the same bound, and *(iii)* it maintains the relative ordering of q-values while compressing extreme outliers. Terminal node q-values remain unchanged as defined in Eq. 6.

Training Objective. With the regularized q-values, we formulate our Tree-GRPO objective as:

$$
\mathcal{J}(\theta) = \mathbb{E}_{\mathbb{T} \sim \mathcal{T}, \mathbf{t}^i \sim \mathbb{T}, (s_j, o_j) \sim \mathbf{t}^i} \frac{1}{|s_j|} \sum_{k=1}^{|s_j|} \min\left(\rho_{j,k}(\theta)\hat{A}_{j,k}, \text{clip}\left(\rho_{j,k}(\theta), 1 - \epsilon_{\text{low}}, 1 + \epsilon_{\text{high}}\right)\hat{A}_{j,k}\right)
\tag{17}
$$

where $\rho_{j,k}(\theta) = \frac{\pi_\theta(a_{j,k}|o_j, a_{j,<k})}{\pi_{\theta_{\text{old}}}(a_{j,k}|o_j, a_{j,<k})}$ is the importance ratio. The parameters ϵ_{high} and ϵ_{low} follow the Clip-Higher strategy of DAPO (Yu et al., 2025), while we also remove the KL regularization term $\mathbb{D}_{\text{KL}}$ to naturally diverge (Luo et al., 2025a; He et al., 2025a). An overlong buffer penalty is imposed to penalize responses that exceed a predefined maximum value of 4096. The advantage function for node s_j in trajectory $\mathbf{t}_i$ is computed using *sequence-level normalization* (Chu et al., 2025):

$$
\hat{A}_{j,k} = q(s_j) - \mu_{\mathbf{t}},
\tag{18}
$$

where $\mu_{\mathbf{t}}$ is the average reward of the terminal nodes $\mathcal{S}_{\text{end}}$ throughout the tree $\mathbb{T}$. This normalization is crucial in practice, particularly for mitigating uncontrolled growth in response length. Notably, Tree-GRPO can be degraded to the vanilla DAPO if we consistently leverage the outcome reward $q(s_{\text{end}})$ as $q(s_j)$ for all intermediate nodes.

Table 1: Performance comparison of 1.5B-scale language models on standard mathematical reasoning benchmarks. We report Pass@1 accuracy with $n = 32$ samples. Results with the best performance are highlighted in bold. All evaluations were conducted on a 128×H100 96G cluster.

Model	AIME24	AIME25	AMC23	MATH	Minerva	Olympiad	Avg
Qwen2.5-Math-1.5B	8.33	6.35	44.06	66.67	18.42	30.74	29.10
Qwen2.5-Math-1.5B-Instruct	10.10	8.85	55.08	74.83	29.32	40.00	36.37
DeepSeek-R1-Distill-Qwen-1.5B	31.15	24.06	72.81	85.01	32.18	51.55	49.46
STILL-3-1.5B	31.46	25.00	75.08	86.24	32.77	53.84	50.73
Qwen2.5-Math-1.5B-Oat-Zero	20.00	10.00	52.50	74.20	26.84	37.78	36.89
Open-RS1-1.5B	30.94	22.60	73.05	84.90	29.92	52.82	49.04
Open-RS2-1.5B	28.96	24.37	73.52	85.06	29.74	52.63	49.05
Open-RS3-1.5B	30.94	24.79	72.50	84.47	29.11	52.25	49.01
DeepScaleR-1.5B	38.54	30.52	80.86	88.79	36.19	58.95	55.64
Nemotron-Research-Reasoning-Qwen-1.5B v1	45.62	33.85	85.70	92.01	39.27	64.56	60.17
Nemotron-Research-Reasoning-Qwen-1.5B v2	51.77	32.92	88.83	92.24	39.75	64.69	61.70
DeepSearch-1.5B	**53.65**	**35.42**	**90.39**	**92.53**	**40.00**	**65.72**	**62.95**

4 EXPERIMENTS

4.1 BENCHMARK PERFORMANCE EVALUATION

Datasets and Base Models. We train DeepSearch based on Nemotron-Research-Reasoning-Qwen-1.5B v2 (Liu et al., 2025a) and employ DeepMath-103K (He et al., 2025c) as the raw dataset. DeepMath-103K is a large-scale mathematical dataset designed with high difficulty, rigorous decontamination against numerous benchmarks. We evaluate DeepSearch against state-of-the-art 1.5B reasoning models on six mathematical benchmarks: AIME 2024/2025, AMC2023, MATH500 (Hendrycks et al., 2021), Minerva (Lewkowycz et al., 2022), and Olympiad (He et al., 2024). More experimental details are described in Appendix B.

Baselines. We compare against recent 1.5B models spanning different paradigms: base models (Qwen2.5-Math variants), RL-trained models (DeepSeek-R1-Distill, STILL-3 (Team, 2025), Open-RS series (Dang & Ngo, 2025), advanced RL methods (DeepScaleR (Luo et al., 2025b), Nemotron variants), and search-based approaches (Qwen2.5-Math-Oat-Zero (Liu et al., 2025b)). Our evaluation methods and results are consistent with Hochlehnert et al. (2025).

Results. Table 1 shows DeepSearch-1.5B achieves 62.95% average accuracy, outperforming all baselines, including the previous best Nemotron-Research-Reasoning-Qwen-1.5B v2 (61.70%). DeepSearch-1.5B demonstrates consistent improvements across all benchmarks, with notable gains on AIME 2024 (53.65% vs 51.77%) and AMC (90.39% vs 88.83%). The 1.25 percentage point improvement over the previous state-of-the-art validates the effectiveness of integrating structured search into RLVR training rather than restricting it to inference only.

4.2 TRAINING EFFICIENCY ANALYSIS

To evaluate the practical viability of DeepSearch, we compare computational costs against extended training approaches that scale purely through additional training steps. As shown in Table 2, extended training shows diminishing returns: 325 additional steps achieve 61.78% accuracy using 326.4 GPU hours, while 1,875 steps plateau at 62.02% despite consuming 1,883.2 GPU hours. This reveals the fundamental limitation of depth-first scaling, where performance gains become marginal as computational investment grows exponentially.

DeepSearch achieves superior results through algorithmic innovation rather than brute-force computation. With only 50 additional training steps, DeepSearch reaches 62.95% accuracy using 330 GPU hours—outperforming the most extensive baseline (1,883.2 hours) while using 5.7× fewer resources. This efficiency stems from a structured search that extracts maximum value from each training step through systematic exploration of diverse solution paths.

Figure 2 illustrates the training dynamics over 20 hours following 3K RLVR training. DAPO exhibits gradual linear improvement with a shallow slope, while DeepSearch demonstrates more efficient

Table 2: Comparison of methods on efficiency and performance, which are trained from DeepSeek-R1-Distill-Qwen-1.5B.

Method	RLVR	Steps	Samples (K)	Time (h)	GPU Hours	Math Score
DeepSeek-R1-Distill-Qwen-1.5B	–	–	–	–	–	49.46
Nemotron-Research-Reasoning-Qwen-1.5B v1	DAPO	2000	–	–	16000	60.10
Nemotron-Research-Reasoning-Qwen-1.5B v2	DAPO	3000	–	–	24000	61.70
Extended Training	DAPO	+325	665.6	20.4	326.4	61.78
Extended Training	DAPO + KL	+785	1607.7	49.3	788.8	62.08
Extended Training	DAPO + KL	+1875	3840.0	117.7	1883.2	62.02
DeepSearch-1.5B	Tree-GRPO	+50	102.4	20.6	330	**62.95**

learning through structured exploration. The superior convergence suggests that RLVR bottlenecks stem from exploration quality rather than insufficient training time.

These results challenge the assumption that scaling RLVR requires proportional computational increases. Compared to the training of Nemotron-Research-Reasoning-Qwen-1.5B v2, DeepSearch-1.5B's $72\times$ efficiency improvement represents a paradigm shift from resource-intensive scaling to algorithmically-driven optimization, demonstrating that systematic exploration outperforms prolonged training for advancing RLVR capabilities.

Figure 2: Average performance (AIME 2024, AIME 2025, and AMC 2023) of **DAPO** and **DeepSearch** after 3K RLVR training. Markers denote evaluations, while dotted lines indicate linear trends.

4.3 SEARCH STRATEGY ABLATION

Table 3 compares our global frontier selection against vanilla UCT under different configurations on 1.2K samples from extremely hard DeepMath-103K problems.

Global vs. Local Selection. Our global frontier selection ($\lambda_1 = 0.4$) reduces iterations by 10.4% ($209.6 \rightarrow 187.7$) and improves trajectory rewards ($-0.82 \rightarrow -0.65$) compared to vanilla UCT, while maintaining similar search depth and entropy. This demonstrates that direct comparison of frontier nodes across the entire tree is more efficient than traditional root-to-leaf UCT traversals.

Depth Bonus Impact. We evaluate three depth bonus functions $D(d(s))$: (i) Logarithmic $\log(d(s) + 1)$ provides minimal improvements, (ii) Linear $d(s)$ achieves the most aggressive efficiency gains with 59% reduction in per-tree time ($1179.6s \rightarrow 480.9s$) and deepest exploration (21.55 depth), but at cost of solution quality (-0.76 reward), (iii) Square root $\sqrt{d(s)/d_T}$ offers the best balance, maintaining search quality (-0.65 reward) with significant computational savings.

Uncertainty Bonus. Adding uncertainty weighting ($\lambda_2 = 0.4$) increases exploration diversity (entropy $1.23 \rightarrow 1.31$) by prioritizing high-uncertainty policy regions, but introduces computational variability (92.5 ± 22.5 iterations).

Configuration Selection. We adopt $\sqrt{d(s)/d_T}$ with $\lambda_1 = 0.4, \lambda_3 = 0.01$ as our default, balancing computational efficiency (189.3 iterations), search quality (-0.65 reward), and stable performance. This configuration eliminates UCT's redundant traversals while maintaining principled exploration through quality potential and depth guidance.

4.4 ALGORITHM EVOLUTION AND COMPONENT CONTRIBUTIONS

To understand the individual contributions of each component, we present a systematic ablation study showing the evolution of our DeepSearch algorithm in Table 4. Starting from the `Nemotron-Research-Reasoning-Qwen-1.5B v2` baseline, we incrementally add components and analyze their impact:

Table 3: Ablation study of different search strategies in DeepSearch. We compare vanilla UCT with our proposed global frontier selection under varying depth bonus functions $D(d(s))$. Reported metrics include search statistics such as average search depth, trajectory entropy, and trajectory reward, together with computational cost measured by the number of iterations, average per-iteration time (in seconds), and per-tree time (in seconds). Results are presented as mean $\pm$ standard deviation.

Method	$D(d(s))$	Search Metrics			Computational Cost		
		Depth	Entropy	Reward	Num. Iter.	Time Per Iter.	Time Per Tree
Vanilla UCT	–	20.11 ± 4.72	1.23 ± 0.29	-0.82 ± 0.57	209.6 ± 14.8	5.63 ± 0.21	1179.6 ± 95.0
Global Frontier Selection							
$\lambda_1 = 0.4$	–	20.28 ± 4.80	1.23 ± 0.29	-0.65 ± 0.76	187.7 ± 16.2	5.76 ± 0.19	1087.7 ± 105.0
$\lambda_1 = 0.4, \lambda_3 = 0.01$	$\log(d(s)+1)$	20.33 ± 4.77	1.23 ± 0.30	-0.65 ± 0.76	185.5 ± 15.9	5.85 ± 0.19	1080.3 ± 102.2
$\lambda_1 = 0.4, \lambda_3 = 0.01$	$d(s)$	21.55 ± 5.13	1.24 ± 0.29	-0.76 ± 0.65	85.7 ± 7.7	5.61 ± 0.12	480.9 ± 41.9
$\lambda_1 = 0.4, \lambda_2 = 0.4, \lambda_3 = 0.01$	$\sqrt{d(s)/d_{\mathcal{T}}}$	20.83 ± 4.71	1.31 ± 0.30	-0.79 ± 0.62	92.5 ± 22.5	5.48 ± 0.13	505.2 ± 114.8
$\boldsymbol{\lambda_1 = 0.4, \lambda_3 = 0.01}$	$\sqrt{d(s)/d_{\mathcal{T}}}$	$\mathbf{20.29 \pm 4.83}$	$\mathbf{1.24 \pm 0.29}$	$\mathbf{-0.65 \pm 0.76}$	$\mathbf{189.3 \pm 14.7}$	$\mathbf{5.66 \pm 0.14}$	$\mathbf{1070.7 \pm 87.3}$

Table 4: Ablation study illustrating the step-by-step evolution of **DeepSearch**. Starting from Vanilla DeepSearch with a simple q-update, we progressively add outcome-reward–based and fine-grained advantages, standard or mean-only normalization, and frontier node selection.

Model / Change	AIME24	AIME25	AMC23	MATH	Minerva	Olympiad	Avg
Nemotron-Research-Reasoning-Qwen-1.5B v2	51.77	32.92	88.83	92.24	39.75	64.69	61.70
+ Vanilla DeepSearch	51.98	34.06	86.64	87.00	37.96	64.00	60.27
+ New q Update & Coarse-grained Token Scores	51.04	**35.73**	86.48	90.66	39.14	65.23	61.38
+ New q Update & Fine-grained Token Scores	50.52	35.52	88.83	91.70	39.71	64.81	61.85
+ Standard Advantages Normalization	52.60	35.00	89.30	92.44	39.29	64.99	62.27
+ Mean-only Advantages Normalization	51.98	**35.73**	89.06	91.88	39.58	65.71	62.32
+ Frontier Selection	**53.65**	35.42	**90.39**	**92.53**	**40.00**	**65.72**	**62.95**

(i) **Vanilla DeepSearch Foundation.** We begin with a basic MCTS integration using a simple q-value update rule:

$$q^{(m)}(s_i) = \begin{cases} q^{(m-1)}(s_i) + \gamma(i,l) \cdot q^{(m)}(s_{\text{end}}) & \text{if } q^{(m-1)}(s_i) \cdot q^{(m)}(s_{\text{end}}) \geq 0, \\ \max\left(q^{(m-1)}(s_i) + \gamma(i,l) \cdot q^{(m)}(s_{\text{end}}), 0\right) & \text{otherwise.} \end{cases}$$

This assigns constant values to nodes on correct reasoning paths but shows limited improvement over the baseline. (ii) **Enhanced Q-Value Updates with Outcome Rewards.** We replace the simple update with our constrained backup rule (Eq. 7) and use outcome-based advantages $\hat{A}_{j,k} = q(s_{\text{end}})$ for all nodes. This provides more stable credit assignment and yields meaningful improvements. (iii) **Fine-Grained Node-Level Advantages.** Moving beyond outcome-only rewards, we assign node-specific advantages $\hat{A}_{j,k} = q(s_j)$ based on each node's individual q-value. This enables more precise credit assignment across different reasoning steps. (iv) **Standard Advantage Normalization.** We implement standard normalization as $\hat{A}_{j,k} = \frac{q(s_j) - \mu_{\mathbf{t}}}{\sigma_{\mathbf{t}} + \varepsilon}$, where $\sigma_{\mathbf{t}}$ is the standard deviation of the rewards of the terminal nodes $\mathcal{S}_{\text{end}}$ throughout the tree $\mathbb{T}$. The constant ε prevents numerical instability when the variance is small. This stabilizes training but introduces variance-based scaling. (v) **Mean-Only Normalization.** We adopt mean-only normalization (Eq. 18). This addresses miscalibration issues in GRPO while maintaining stable advantage scaling. (Bereket & Leskovec, 2025). (vi) **Global Frontier Selection.** Finally, we integrate our novel frontier selection strategy (Eq. 9), which prioritizes promising expansion candidates across the entire search tree rather than following traditional root-to-leaf UCT-like traversals.

The results demonstrate that each component contributes meaningfully to the final performance, with frontier selection providing the largest single improvement. The cumulative effect shows that systematic exploration and fine-grained credit assignment are both essential for maximizing the benefits of search-augmented RLVR.

5 CONCLUSION

We introduced DeepSearch, which integrates Monte Carlo Tree Search directly into RLVR training to address exploration bottlenecks that cause performance plateaus. Our framework features global frontier selection, entropy-based guidance, and adaptive replay buffers with the Tree-GRPO objective for fine-grained credit assignment. DeepSearch achieves 62.95% average accuracy on

mathematical reasoning benchmarks, establishing a new state-of-the-art for 1.5B models with 1.25 percentage point improvement over previous best methods while using 5.7× fewer GPU hours. This demonstrates that systematic exploration during training is more effective than prolonged computation, shifting the paradigm from scaling training depth to scaling training breadth through algorithmic innovation.

ETHICS STATEMENT

This work advances automated mathematical reasoning through algorithmic innovation without exaggerated capability claims. We commit to releasing complete implementation details for reproducibility and transparency. While enhanced reasoning capabilities could benefit education and scientific computing, we acknowledge potential dual-use concerns, though mathematical domains with verifiable correctness limit harmful applications. Our approach reduces computational requirements (330 vs 1883 GPU hours) compared to extended training, potentially decreasing environmental impact. We will make our implementation publicly available to support open science and broader community engagement.

REFERENCES

Lei Bai, Zhongrui Cai, Maosong Cao, Weihan Cao, Chiyu Chen, Haojiong Chen, Kai Chen, Pengcheng Chen, Ying Chen, Yongkang Chen, et al. Intern-s1: A scientific multimodal foundation model. *arXiv preprint arXiv:2508.15763*, 2025.

Michael Bereket and Jure Leskovec. Uncalibrated reasoning: Grpo induces overconfidence for stochastic outcomes. *arXiv preprint arXiv:2508.11800*, 2025.

Graeme Best, Oliver M Cliff, Timothy Patten, Ramgopal R Mettu, and Robert Fitch. Dec-mcts: Decentralized planning for multi-robot active perception. *The International Journal of Robotics Research*, 38(2-3):316–337, 2019.

Zhenni Bi, Kai Han, Chuanjian Liu, Yehui Tang, and Yunhe Wang. Forest-of-thought: Scaling test-time compute for enhancing llm reasoning. *arXiv preprint arXiv:2412.09078*, 2024.

Guoxin Chen, Minpeng Liao, Chengxi Li, and Kai Fan. Alphamath almost zero: process supervision without process. *Advances in Neural Information Processing Systems*, 37:27689–27724, 2024.

Xiangxiang Chu, Hailang Huang, Xiao Zhang, Fei Wei, and Yong Wang. Gpg: A simple and strong reinforcement learning baseline for model reasoning. *arXiv preprint arXiv:2504.02546*, 2025.

Tuan Dam, Georgia Chalvatzaki, Jan Peters, and Joni Pajarinen. Monte-carlo robot path planning. *IEEE Robotics and Automation Letters*, 7(4):11213–11220, 2022.

Quy-Anh Dang and Chris Ngo. Reinforcement learning for reasoning in small llms: What works and what doesn't. *arXiv preprint arXiv:2503.16219*, 2025.

DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco J R. Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, et al. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930):47–53, 2022.

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

Anisha Gunjal, Anthony Wang, Elaine Lau, Vaskar Nath, Bing Liu, and Sean Hendryx. Rubrics as rewards: Reinforcement learning beyond verifiable domains. *arXiv preprint arXiv:2507.17746*, 2025.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, et al. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. *arXiv preprint arXiv:2402.14008*, 2024.

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

Shenghua He, Tian Xia, Xuan Zhou, and Hui Wei. Response-level rewards are all you need for online reinforcement learning in llms: A mathematical perspective. *arXiv preprint arXiv:2506.02553*, 2025b.

Zhiwei He, Tian Liang, Jiahao Xu, Qiuzhi Liu, Xingyu Chen, Yue Wang, Linfeng Song, Dian Yu, Zhenwen Liang, Wenxuan Wang, et al. Deepmath-103k: A large-scale, challenging, de-contaminated, and verifiable mathematical dataset for advancing reasoning. *arXiv preprint arXiv:2504.11456*, 2025c.

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

Andreas Hochlehnert, Hardik Bhatnagar, Vishaal Udandarao, Samuel Albanie, Ameya Prabhu, and Matthias Bethge. A sober look at progress in language model reasoning: Pitfalls and paths to reproducibility. *arXiv preprint arXiv:2504.07086*, 2025.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

Marco Kemmerling, Daniel Lütticke, and Robert H Schmitt. Beyond games: a systematic review of neural monte carlo tree search applications. *arXiv preprint arXiv:2303.08060*, 2023.

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

Guillaume Lample, Timothee Lacroix, Marie-Anne Lachaux, Aurelien Rodriguez, Amaury Hayat, Thibaut Lavril, Gabriel Ebner, and Xavier Martinet. Hypertree proof search for neural theorem proving. *Advances in neural information processing systems*, 35:26337–26349, 2022.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. Solving quantitative reasoning problems with language models. *Advances in neural information processing systems*, 35:3843–3857, 2022.

Yifei Li, Zeqi Lin, Shizhuo Zhang, Qiang Fu, Bei Chen, Jian-Guang Lou, and Weizhu Chen. Making language models better reasoners with step-aware verifier. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5315–5333, 2023.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2023.

Mingjie Liu, Shizhe Diao, Ximing Lu, Jian Hu, Xin Dong, Yejin Choi, Jan Kautz, and Yi Dong. Prorl: Prolonged reinforcement learning expands reasoning boundaries in large language models. *arXiv preprint arXiv:2505.24864*, 2025a.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding r1-zero-like training: A critical perspective. *arXiv preprint arXiv:2503.20783*, 2025b.

Zihe Liu, Jiashun Liu, Yancheng He, Weixun Wang, Jiaheng Liu, Ling Pan, Xinyu Hu, Shaopan Xiong, Ju Huang, Jian Hu, et al. Part i: Tricks or traps? a deep dive into rl for llm reasoning. *arXiv preprint arXiv:2508.08221*, 2025c.

Michael Luo, Sijun Tan, Roy Huang, Ameen Patel, Alpay Ariyak, Qingyang Wu, Xiaoxiang Shi, Rachel Xin, Colin Cai, Maurice Weber, et al. Deepcoder: A fully open-source 14b coder at o3-mini level. *Notion Blog*, 2025a.

Michael Luo, Sijun Tan, Justin Wong, Xiaoxiang Shi, William Y. Tang, Manan Roongta, Colin Cai, Jeffrey Luo, Li Erran Li, Raluca Ada Popa, and Ion Stoica. Deepscaler: Surpassing o1-preview with a 1.5b model by scaling rl, 2025b. Notion Blog.

Chengqi Lyu, Songyang Gao, Yuzhe Gu, Wenwei Zhang, Jianfei Gao, Kuikun Liu, Ziyi Wang, Shuaibin Li, Qian Zhao, Haian Huang, et al. Exploring the limit of outcome reward for learning mathematical reasoning. *arXiv preprint arXiv:2502.06781*, 2025.

Nicholas Metropolis and Stanislaw Ulam. The monte carlo method. *Journal of the American statistical association*, 44(247):335–341, 1949.

Yelisey Pitanov, Alexey Skrynnik, Anton Andreychuk, Konstantin Yakovlev, and Aleksandr Panov. Monte-carlo tree search for multi-agent pathfinding: Preliminary results. In *International Conference on Hybrid Artificial Intelligence Systems*, pp. 649–660. Springer, 2023.

Zhenting Qi, Mingyuan Ma, Jiahang Xu, Li Lyna Zhang, Fan Yang, and Mao Yang. Mutual reasoning makes smaller llms stronger problem-solvers. *arXiv preprint arXiv:2408.06195*, 2024.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models. 2024.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*, 2024.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

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

Yi Su, Dian Yu, Linfeng Song, Juntao Li, Haitao Mi, Zhaopeng Tu, Min Zhang, and Dong Yu. Crossing the reward bridge: Expanding rl with verifiable rewards across diverse domains. *arXiv preprint arXiv:2503.23829*, 2025.

RUCAIBox STILL Team. Still-3-1.5b-preview: Enhancing slow thinking abilities of small models through reinforcement learning. 2025. URL `https://github.com/RUCAIBox/Slow_Thinking_with_LLMs`.

Harshil Vagadia, Mudit Chopra, Abhinav Barnawal, Tamajit Banerjee, Shreshth Tuli, Souvik Chakraborty, and Rohan Paul. Phyplan: Compositional and adaptive physical task reasoning with physics-informed skill networks for robot manipulators. *arXiv preprint arXiv:2402.15767*, 2024.

Zhongwei Wan, Zhihao Dou, Che Liu, Yu Zhang, Dongfei Cui, Qinjian Zhao, Hui Shen, Jing Xiong, Yi Xin, Yifan Jiang, et al. Srpo: Enhancing multimodal llm reasoning via reflection-aware reinforcement learning. *arXiv preprint arXiv:2506.01713*, 2025.

Peisong Wang, Ruotian Ma, Bang Zhang, Xingyu Chen, Zhiwei He, Kang Luo, Qingsong Lv, Qingxuan Jiang, Zheng Xie, Shanyi Wang, et al. Rlver: Reinforcement learning with verifiable emotion rewards for empathetic agents. *arXiv preprint arXiv:2507.03112*, 2025a.

Peiyi Wang, Lei Li, Zhihong Shao, RX Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. *arXiv preprint arXiv:2312.08935*, 2023.

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

Yiping Wang, Qing Yang, Zhiyuan Zeng, Liliang Ren, Liyuan Liu, Baolin Peng, Hao Cheng, Xuehai He, Kuan Wang, Jianfeng Gao, et al. Reinforcement learning for reasoning in large language models with one training example. *arXiv preprint arXiv:2504.20571*, 2025b.

Fang Wu, Weihao Xuan, Ximing Lu, Zaid Harchaoui, and Yejin Choi. The invisible leash: Why rlvr may not escape its origin. *arXiv preprint arXiv:2507.14843*, 2025.

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. Inference scaling laws: An empirical analysis of compute-optimal inference for problem-solving with language models. *arXiv preprint arXiv:2408.00724*, 2024.

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, Keming Lu, Mingfeng Xue, Runji Lin, Tianyu Liu, Xingzhang Ren, and Zhenru Zhang. Qwen2.5-math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*, 2024.

Feiyu Yang. An integrated framework integrating monte carlo tree search and supervised learning for train timetabling problem. *arXiv preprint arXiv:2311.00971*, 2023.

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

Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, and Yang Gao. Mastering atari games with limited data. *Advances in neural information processing systems*, 34:25476–25488, 2021.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model? *arXiv preprint arXiv:2504.13837*, 2025.

Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. Rest-mcts*: Llm self-training via process reward guided tree search. *Advances in Neural Information Processing Systems*, 37:64735–64772, 2024a.

Di Zhang, Xiaoshui Huang, Dongzhan Zhou, Yuqiang Li, and Wanli Ouyang. Accessing gpt-4 level mathematical olympiad solutions via monte carlo tree self-refine with llama-3 8b. *arXiv preprint arXiv:2406.07394*, 2024b.

Di Zhang, Jianbo Wu, Jingdi Lei, Tong Che, Jiatong Li, Tong Xie, Xiaoshui Huang, Shufei Zhang, Marco Pavone, Yuqiang Li, et al. Llama-berry: Pairwise optimization for o1-like olympiad-level mathematical reasoning. *arXiv preprint arXiv:2410.02884*, 2024c.

A   RELATED WORKS

Search-based reasoning.   Structured search has become a standard strategy for scaling test-time computation in LLMs (Snell et al., 2024; Wu et al., 2024; Zhang et al., 2024c), with diverse methods including tree-based (Yao et al., 2023; Zhang et al., 2024b; Qi et al., 2024) and random sampling approaches (Wang et al., 2022). More recently, search-based reasoning has evolved into sophisticated frameworks that integrate three core components: policy models for generating reasoning steps, reward models for evaluative feedback, and search algorithms for exploring solution spaces. Drawing inspiration from game-playing systems like AlphaGo, recent works have explored Monte Carlo Tree Search (MCTS) and beam search to guide LLMs through structured reasoning processes (Chen et al., 2024; Zhang et al., 2024a;c), particularly following OpenAI's o1 model release (Jaech et al., 2024). These frameworks enable exploration of multiple solution paths during inference, trading computational resources for improved accuracy on challenging tasks such as mathematical reasoning. Key design considerations include outcome-supervised versus process-supervised reward models, discriminative versus generative reward architectures, and search strategies ranging from local selection to global exploration (Lightman et al., 2023; Wang et al., 2023). However, despite their effectiveness, most current methods restrict search to inference without integrating exploration signals into training, leaving the potential for joint optimization of search and learning largely unexplored.

Reinforcement learning from verifiable rewards.   RLVR has emerged as a transformative approach for aligning and enhancing LLMs by addressing critical challenges across instruction following (Su et al., 2025; Gunjal et al., 2025), ethical alignment (Wang et al., 2025a), and reasoning capabilities (Wang et al., 2025b). Recent extensions (Guo et al., 2025; Yu et al., 2025; Wan et al., 2025) have improved training stability and efficiency by incorporating critic-free optimization, dynamic sampling, and adaptive weighting mechanisms. While these approaches demonstrate the significant promise of verifiable rewards, they predominantly rely on direct rollouts, which can constrain systematic exploration of the solution space (Wu et al., 2025; Yue et al., 2025).

Monte-Carlo Tree Search.   MCTS is a powerful search paradigm for complex decision-making problems that has been extensively explored across diverse fields, including games (Silver et al., 2016; Ye et al., 2021), robotics (Best et al., 2019; Dam et al., 2022), theorem proving (Lample et al., 2022), and matrix multiplication (Fawzi et al., 2022). Early work such as AlphaGo (Silver et al., 2016) successfully integrated MCTS with deep learning (Kemmerling et al., 2023), achieving superhuman performance in board and video games (Ye et al., 2021). More recently, MCTS has been applied to path finding and train timetabling problems (Pitanov et al., 2023; Yang, 2023), while Vagadia et al. (2024) integrated MCTS into physics-informed planning networks for robot control. Despite the demonstrated potential of MCTS for heuristic exploration, it remains unclear how to effectively employ it during RLVR training.

B   EXPERIMENTAL DETAILS

This section provides comprehensive details of our experimental setup, including system implementation, training configurations, MCTS parameters, optimization strategies, and evaluation protocols used in our DeepSearch framework.

B.1   TRAINING DATA AND CONFIGURATION

We implement our DeepSearch system using the veRL framework (Sheng et al., 2024), conducting all training experiments on a distributed setup across 16 NVIDIA H100 GPUs with 96GB of memory. The policy model is initialized with Nemotron-Research-Reasoning-Qwen-1.5B v2 (Liu et al., 2025a) (updated July 23rd). To ensure a fair comparison on a well-aligned policy, we additionally conduct DAPO-based extended training on the Nemotron-Research-Reasoning-Qwen-1.5B-v2 initialization, using the same training configuration as DeepSearch.

Our training methodology employs the DeepMath-103K (He et al., 2025c) dataset as D_{train}, implementing a DeepScaleR-style prompt template that instructs the model to "Let's think step by step and output the final answer within \boxed{}." To manage computational constraints, we apply prompt truncation from the left with a maximum prompt length of 2,048 tokens and limit response

generation to 16,384 tokens. The training process utilizes a global batch size of 256 samples, implemented through the DAPO-style Dynamic Batching strategy (Yu et al., 2025) to optimize memory utilization and training efficiency.

B.2 MONTE CARLO TREE SEARCH IMPLEMENTATION

Our MCTS implementation incorporates several strategic design choices to balance search efficiency and solution quality. The exploration coefficient (λ) for UCT Local Selection is set to 2.0, providing an optimal exploration-exploitation trade-off for mathematical reasoning tasks. The search architecture operates with a maximum depth of 64 levels, where each node is allocated 256 tokens and expands 8 children during the expansion phase. For entropy-based selection, we estimate the average trajectory entropy using only tokens that appear in the response instead of the entire per-position vocabulary for computational efficiency.

To enhance search effectiveness, the system employs Global Frontier Selection for backtrace operations and applies a square root function for depth-based bonuses, encouraging deeper exploration when beneficial. The global λ_3 parameter is configured to 0.01 for our frontier priority scoring, while an overlong buffer of 4,096 tokens with a penalty factor of 1.0 accommodates lengthy reasoning chains typical in complex mathematical problems.

B.3 ADVANTAGE ESTIMATION AND OPTIMIZATION

For advantage estimation, we implement the Grouped Relative Policy Optimization (GRPO) (Shao et al., 2024) estimator with sibling mean normalization to ensure stable learning dynamics. The Q-value soft clipping mechanism operates at a temperature of 1.0 with the maximum q-value magnitude set to 1.0, while incomplete trajectories receive a penalty score of -1.0 to discourage premature termination. Standard deviation normalization is disabled to prevent numerical instability during training.

The actor model optimization employs AdamW with a conservative learning rate of 1×10^{-6} and 10 warmup steps, combined with weight decay of 0.1 and gradient clipping at 1.0 for stable convergence. We follow the Clip-Higher strategy in DAPO (Yu et al., 2025) and set the lower and higher clipping range to 0.2 and 0.28 with a ratio coefficient of 10.0. Training proceeds with mini-batches of 32 samples per policy update using token-mean loss aggregation, while dynamic batch sizing accommodates up to 18,432 tokens per GPU. The entropy coefficient is set to 0 for pure exploitation, and KL divergence loss is disabled to maximize performance on the target mathematical reasoning tasks.

B.4 SAMPLING AND REWARD CONFIGURATION

During rollout generation, we configure sampling parameters with a high temperature of 1.0 and top_p of 1.0, while disabling top_k filtering to maintain diverse response generation. The system generates 8 rollouts per prompt, aligning with the expansion width parameter, within a context length of 18,432 tokens. This configuration ensures comprehensive exploration of the solution space while maintaining computational feasibility. During evaluation, we uniformly use a low temperature of 0.6 and top_p of 0.95.

Our reward system implements a custom mathematical scoring function based on (`compute_score`) from the `math_dapo.py` module, designed to evaluate mathematical reasoning accuracy. We extract the final boxed answer by locating the last occurrence of \boxed{} in the trajectory and apply the same text-normalization logic as veRL's DAPO recipe to both prediction and ground-truth. The reward mechanism handles responses up to 16,384 tokens, following ProRL (Liu et al., 2025a) and ensuring consistent evaluation across varying response lengths.

B.5 TRAINING PROTOCOL

The complete training protocol spans 100 steps with model checkpointing performed every 5 steps. This frequent checkpointing strategy ensures robust model preservation and enables detailed analysis of learning progression throughout the training process.

C PSEUDOCODE OF DEEPSEARCH

Algorithm 1 presents the complete DeepSearch framework, integrating MCTS-based exploration with adaptive training and replay buffer management. The algorithm operates through iterative refinement, progressively focusing computational resources on challenging problems while preserving solved solutions through intelligent caching. This integrated approach focuses on training-time exploration, enabling models to learn from both correct solutions and systematic exploration processes rather than relying solely on outcome-based supervision.

LIMITATIONS AND FUTURE WORK

A critical next step involves extending DeepSearch beyond mathematical reasoning to domains with different verification mechanisms. This includes developing approximate verifiers for subjective tasks, exploring human-in-the-loop validation for complex reasoning chains, and investigating transfer learning approaches that leverage mathematical reasoning capabilities for broader problem-solving tasks. Research into domain-agnostic reward functions and verification strategies could significantly expand the framework's applicability.

Algorithm 1 DeepSearch with Global Frontier Selection and Iterative Filtering

Require: Initial policy $\pi_{\theta^{(0)}}$, training set $\mathcal{D}_{\text{train}}$, verifier $\mathcal{V}$, filtering threshold δ

1: Initialize $\mathcal{D}_{\text{hard}}^{(0)} \leftarrow \{x \in \mathcal{D}_{\text{train}} \mid \texttt{Pass1@K}(x, \pi_{\theta^{(0)}}) < \delta^{(0)}\}$, $\mathcal{R}^{(0)} = \emptyset$
2: **for** training iteration $i = 0, 1, 2, \ldots$ **do**
3:      Initialize training trajectories $\mathcal{T}_{\text{train}}^{(i)} \leftarrow \emptyset$
4:      **for** each batch $\mathcal{B}^{(i)} \in \mathcal{D}_{\text{hard}}^{(i)}$ **do**
5:          **for** each problem $x \in \mathcal{B}^{(i)}$ **do**
6:              **if** $(x, \mathbf{t}_{\text{cached}}) \in \mathcal{R}^{(i)}$ **then** $\triangleright$ Use cached solution
7:                  $\mathcal{T}_x \leftarrow \{\mathbf{t}_{\text{cached}}\} \cup \text{DirectRollouts}(x, \beta)$
8:                  $\mathcal{T}_{\text{train}}^{(i)} \leftarrow \mathcal{T}_{\text{train}}^{(i)} \cup \mathcal{T}_x$
9:              **else** $\triangleright$ Apply full MCTS search
10:                  **MCTS Search:**
11:                  Initialize search tree $\mathcal{T}$ with root node x
12:                  **for** rollout iteration $k = 1, 2, \ldots$ **do**
13:                      **if** $k = 1$ **then** $\triangleright$ Initial expansion from root
14:                          Select root node $s^* = x$ for expansion
15:                      **else**
16:                          **Global Frontier Selection:**
17:                          Compute frontier set $\mathcal{F} = \{s \in \mathcal{T} \mid \xi(s) = 0, s \notin \mathcal{S}_{\text{end}}, d(s) < d_{\mathcal{T}}\}$
18:                          Compute frontier priority scores (Eq. 10)
19:                          Select node $s^* = \arg\max_{s \in \mathcal{F}} F(s)$ for expansion
20:                      **end if**
21:                      **Local Expansion with UCT Selection:**
22:                      Generate n candidates $\{s_j\}_{j=1}^n \sim \pi_\theta(\cdot \mid o_{s^*})$ from s^*
23:                      Continue expansion until terminal nodes $\mathcal{S}_{\text{end}}^{(k)}$ are reached
24:                      **Evaluation with Entropy-based Guidance**
25:                      Partition: $\mathcal{S}_{\text{correct}}^{(k)} = \{s \in \mathcal{S}_{\text{end}}^{(k)} \mid \mathcal{V}(s) = 1\}$, $\mathcal{S}_{\text{incorrect}}^{(k)} = \{s \in \mathcal{S}_{\text{end}}^{(k)} \mid \mathcal{V}(s) = 0\}$
26:                      **if** $|\mathcal{S}_{\text{correct}}^{(k)}| \geq 1$ **then**
27:                          Extract trajectories $\mathbb{T}(x)$ from search tree $\mathcal{T}$
28:                          $\mathcal{T}_{\text{train}}^{(i)} \leftarrow \mathcal{T}_{\text{train}}^{(i)} \cup \mathbb{T}(x)$
29:                      **else**
30:                          Select most confident negative: $s_{\text{neg}}^* = \arg\min_{s \in \mathcal{S}_{\text{incorrect}}^{(k)}} \bar{H}(\mathbf{t}(s))$
31:                      **end if**
32:                      **Heuristic Score Backup:**
33:                      Select trajectory $\mathbf{t}^*$ (correct solution or $\mathbf{t}(s_{\text{neg}}^*)$)
34:                      Assign terminal rewards (Eq. 6)
35:                      **for** each node s_j in $\mathbf{t}^*$ **do**
36:                          Update Q-values using constrained backup rule (Eq. 7)
37:                      **end for**
38:                  **end for**
39:              **end if**
40:              **Replay Buffer Update:**
41:              **if** MCTS found correct solutions but $\texttt{Pass1@K}(x, \pi_{\theta^{(i)}}) < \delta^{(i)}$ **then**
42:                  Add $(x, \mathbf{t}_{\text{correct}})$ to $\mathcal{R}^{(i+1)}$ for any correct $\mathbf{t}_{\text{correct}} \in \mathbb{T}(x)$
43:              **end if**
44:          **end for**
45:          **Policy Update:**
46:          Update policy $\pi_{\theta^{(i+1)}}$ using Tree-GRPO objective on $\mathcal{T}_{\text{train}}^{(i)}$ (Eq. 16 and Eq. 17)
47:      **end for**
48:      Re-evaluate and filter: $\mathcal{D}_{\text{hard}}^{(i+1)} = \{x \in \mathcal{D}_{\text{hard}}^{(i)} \mid \texttt{Pass1@K}(x, \pi_{\theta^{(i+1)}}) < \delta^{(i+1)}\}$
49: **end for**