`LiteSearch`: Efficient Tree Search with Dynamic Exploration Budget for Math Reasoning

Ante Wang[1,3], Linfeng Song[2*], Ye Tian[2], Baolin Peng[2], Dian Yu[2], Haitao Mi[2], Jinsong Su[1,3,4*], Dong Yu[2]

[1]School of Informatics, Xiamen University, China
[2]Tencent AI Lab, Bellevue, WA
[3]Shanghai Artificial Intelligence Laboratory, China
[4]Key Laboratory of Multimedia Trusted Perception and Efficient Computing, Ministry of Education of China, Xiamen University, China
wangante@stu.xmu.edu.cn, lfsong@global.tencent.com, jssu@xmu.edu.cn

Abstract

Recent research suggests that tree search algorithms (e.g. Monte Carlo Tree Search) can dramatically boost LLM performance on complex mathematical reasoning tasks. However, they often require more than 10 times the computational resources of greedy decoding due to wasteful search strategies, making them difficult to be deployed in practical applications. This study introduces a novel guided tree search algorithm with a goal-directed heuristic function and node-level exploration budget (maximum number of children) calculation to tackle this issue. By considering the search progress towards the final answer (history) and the guidance from a value network (future) trained without any step-wise annotations, our algorithm iteratively selects the most promising tree node before expanding it within the boundaries of the allocated computational budget. Experiments conducted on the GSM8K, TabMWP, and MATH datasets demonstrate that our method not only offers competitive performance but also enjoys significantly lower computational costs compared to baseline methods.

1 Introduction

Mathematical reasoning tasks (Amini et al. 2019; Cobbe et al. 2021; Hendrycks et al. 2021; Lu et al. 2022) have long been acknowledged as challenging. These tasks require transforming a question into a sequence of reasoning steps, which are subsequently executed to derive the correct answer. Recently, large language models (LLMs, Achiam et al. 2023; Touvron et al. 2023; Jiang et al. 2024) have demonstrated remarkable potential in addressing them. A pivotal approach is the employment of Chain-of-Thought (CoT) prompting (Wei et al. 2022; Kojima et al. 2022), which prompts LLMs to break down a question solution into a sequence of reasoning steps before reaching an answer.

Despite their impressive capabilities, LLMs still face challenges when tackling problems with increasing reasoning steps due to the nature of auto-regressive decoding. This can be analogous to the "System 1" mode of thought in psychology (Daniel 2017), which is characterized by fast, intuitive, but error-prone thinking. Much of recent work has focused on enhancing the "System 1" capability of LLMs by prompt-engineering, such as hierarchical prompting (Suzgun and Kalai 2024; Zeng et al. 2023) and automatic prompt refinement (Madaan et al. 2024; Yang et al. 2024; Zhou et al. 2024). On the other hand, growing research attention is being paid to promote the "System 2" mode of thought (Daniel 2017) for LLMs. It is characterized by deliberative thinking steps with back-and-forth refinements, which can be essential for solving complex math reasoning tasks. Particularly, prior efforts have studied enhancing LLMs both at inference time and through self-improvement using tree search algo-

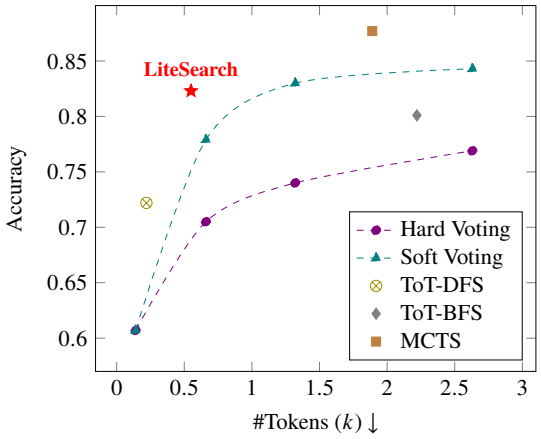

Figure 1: Comparison among ours and typical methods on GSM8K, where DFS, BFS, MCTS[1], and LiteSearch are only guided by the same value network. Hard Voting is also known as Self-Consistency (Wang et al. 2022) and Soft Voting additionally weights each trajectory using its value score. We measure the number of generated tokens (*#Tokens (k)*) by the LLM as computation costs.

[*] Corresponding Authors

[1]The MCTS implemented here does not utilize simulation for fair comparison and efficiency.

rithms, such as Breath-first Search (BFS, Khalifa et al. 2023; Yao et al. 2024; Xie et al. 2024; Zhu et al. 2024) and Monte Carlo Tree Search (MCTS, Feng et al. 2023; Tian et al. 2024; Zhang et al. 2024; Wang et al. 2024c).

However, these approaches often necessitate extensive manual labeling (Ma et al. 2023; Kang et al. 2024), making them difficult to be adopted in different settings. Moreover, they are computationally intensive, especially when tackling problems that require numerous logical steps (Xie et al. 2024). This is because these methods ineffectively manage the expansion budget (the number of nodes to expand) throughout the search process. As a typical example, BFS adopts a constant budget size throughout the search process, overlooking the fact that some tree nodes do not require much exploration. Some MCTS approaches (Tian et al. 2024) take adaptive budget sizes based on the importance of each node. Nevertheless, they still require a large number of simulations or rollouts for accurate statistics to make decisions, and they overlook other important information, such as the depth (progress) of each node. As a result, there is a pressing need to develop more efficient and adaptable methods for enhancing LLMs' "System 2" reasoning capabilities to effectively handle complex reasoning tasks.

In this study, we introduce a guided tree search algorithm with dynamic node-level exploration budget calculation, aiming to maintain the performance at a moderate cost. Concretely, we utilize the value score from a value network (Tian et al. 2024) and the progress toward the final answer as heuristic guidance to *select* the most promising node from the search tree. Then, we *expand* this node within a dynamically computed budget size, effectively navigating the balance between exploration and exploitation for guided tree search. We continue iterating operations of selection and expansion until the resulting trajectory either meets the expected quality score or surpasses the maximum number of iterations. Notably, the computational budget for each node is inversely correlated to its value score. This is inspired by the observation that nodes with higher value scores are more likely to yield the correct solutions upon expansion, hence we allocate fewer computational resources for them to prevent unnecessary computation and vice versa. This not only promotes efficient exploitation, facilitating a faster convergence to the final answer, but also guarantees sufficient exploration to cover enough state space for maintaining performance.

We conduct experiments on the popular GSM8K (Cobbe et al. 2021), TabMWP (Lu et al. 2022), and MATH (Hendrycks et al. 2021). Results show that our method offer competitive performance but significantly less computation costs compared to other baselines. Detailed analyses confirm the usefulness of each component and provide more practical options for various settings. Additionally, we identify the limitations of this research line and suggest possible ways to tackle them.

2 Related Work

Thanks to the strong capabilities of LLMs, significant advancements have been made in mathematical reasoning tasks, surpassing traditional approaches that rely on semantic parsing (Matsuzaki et al. 2017; Hopkins et al. 2017) or Abstract Syntax Tree (AST) decoding (Li et al. 2019; Qin et al. 2021).

Some studies improved the reasoning capabilities of LLMs through further training. These efforts involve either manually annotating or automatically generating feasible and challenging problems to fine-tune LLMs (Luo et al. 2023; Yu et al. 2023; Liu and Yao 2024; Huang et al. 2024), as well as devising sophisticated techniques, such as reinforcement learning, for efficient training (Luo et al. 2023; Wang et al. 2023; Lightman et al. 2023; Chen et al. 2024).

Another line of research focused on inference-time improvement. Except for prompting engineering (Kojima et al. 2022; Liu et al. 2023) and the popular self-consistency (Wang et al. 2022, 2024a,b), most of these studies treat this task as a tree search problem and investigate various searching algorithms. Yao et al. (2024) were the first to introduce Tree-of-Thought (ToT), incorporating Depth-first Search (DFS) and Breath-first Search (BFS) to address reasoning problems. Some researchers (Khalifa et al. 2023; Zhu et al. 2024; Xie et al. 2024) applied step-wise Beam Search to math problems, which operates similarly to BFS under certain parameter conditions. To guide the search process, these studies either directly prompt LLMs to evaluate the quality of each step (Yao et al. 2024; Xie et al. 2024), or train a verifier on corresponding datasets to achieve better performance (Khalifa et al. 2023; Zhu et al. 2024).

Later studies delved into other sophisticated search algorithms, such as Monte Carlo Tree Search (MCTS, Tian et al. 2024; Zhang et al. 2024; Wang et al. 2024c), A* (Ma et al. 2023), and Levin Tree Search (Kang et al. 2024). Nonetheless, these approaches necessitate more robust verifiers to steer the search procedure. Concretely, Tian et al. (2024) utilized a blend of the value function, Process-supervised Reward Model (PRM), and Outcome-supervised Reward Model (ORM). Ma et al. (2023) and Kang et al. (2024) train their PRM models on PRM800K (Lightman et al. 2023), which offers manual annotations for $800k$ reasoning steps of problems from MATH (Hendrycks et al. 2021).

This study also follows the same research line, yet it concentrates on developing an efficient algorithm to decrease computation costs while maintaining performance. Besides, we employ a naive but more practical value network as the verifier, which is trained solely with the final answer labels as distant supervision.

3 Our Method

In this section, we first introduce LiteSearch (§3.1), an efficient tree search algorithm designed to maintain model performance at minimum computation costs. Then, we describe the value network (§3.2), which is easy to obtain and effectively guides the tree search process.

3.1 Guided Tree Search Algorithm

Taking the solving of each math reasoning question q as a tree search problem, we initialize the root of the search tree with question q, while the other tree nodes represent reasoning steps (e.g., s_i) generated by an LLM (denoted as

Algorithm 1: LiteSearch

Input: question q
Parameter: policy π, value network v^π, threshold ε, maximum iteration number N
Output: solution $\hat{y}$

Initialize tree $\mathcal{T}$ with q as the root
$i \leftarrow 0, \hat{y} \leftarrow null$
while $i < N$ **do**
  Select the node s' from $\mathcal{T}$ using Eq. 2
  Expand s' to obtain its child nodes C under the budget constraint b computed by Eq. 3
  for $c \in C$ **do**
    $\mathbf{s} \leftarrow$ return_trajectory$(\mathcal{T}, c)$
    if is_terminal$(\mathbf{s})$ and $v^\pi(\mathbf{s}) > v^\pi(\hat{y})$ **then**
      $\hat{y} \leftarrow \mathbf{s}$
    end if
  end for
  if $v^\pi(\hat{y}) > \varepsilon$ **then**
    break
  end if
  $i \leftarrow i + 1$
end while
return $\hat{y}$

policy π). Concretely, we treat an (incomplete) trajectory $q, s_1, ..., s_i$ as the state $\mathbf{s}_i$.[2] Then, a next step can be sampled from the LLM which consumes $\mathbf{s}_i$ as follows:

$$s_{i+1} \sim \pi(\mathcal{D}, \mathbf{s}_i), \tag{1}$$

where $\mathcal{D}$ is the in-context demonstrations consisting of question-solution pairs or task-specific instruction.

As shown in Alg. 1, our algorithm mainly comprises an iterative process of *Selection* and *Expansion* operations. During each iteration, we first *select* the most promising node, and then *expand* it within the constraints of the computational budget. Both operations are guided by a value network v^π (§3.2). The algorithm terminates when the generated answers meet the expected value threshold ε or the number of iterations reaches the limit N.

Selection We mainly select the tree node with the highest value for expansion. Besides, we introduce a *progress term*, denoted as $p(\mathbf{s})$, which quantifies the advancement of a state $\mathbf{s}$ towards the goal within the search trajectory. By incorporating this term, we prioritize the exploration of nodes that are expected to lead more rapidly toward the final answer. Formally, for each iteration, the next node to explore is selected via

$$s' = \max_{s_i}(v^\pi(\mathbf{s}_i) + \lambda p(\mathbf{s}_i)), \tag{2}$$

where s' denotes the selected node, and λ is introduced to regulate the impact of the progress term.

However, it is non-trivial to estimate the progress of a state. To deal with this issue, we introduce an empirical approach based on the trajectory of greedy decoding. Specifically, we compute the progress term by comparing the number of tokens or steps from a given state to those of the corresponding greedy decoding. For example, when using the step number as the metric, the progress of a state with d steps is $\min(d/\hat{d}, 1)$, where $\hat{d}$ denotes the total number of steps in the trajectory of greedy decoding.

Expansion During the expansion phase, we aim to balance the exploitation and exploration by effectively managing the computation budget allocated to the selected node. Intuitively, an appropriate budget size can promote efficient exploitation, facilitating a faster convergence to the final answer, while also guaranteeing sufficient exploration to cover enough state space for reducing uncertainty. In line with this spirit, we further explore two strategies preferring either exploitation or exploration: *Incremental Expansion* and *Batch Expansion*.

Budget Computaton We define the allocated budget for a node (corresponding to $\mathbf{s}$) as the maximum number of its children, denoted as b, which primarily depends on the value $v^\pi(\mathbf{s})$ and depth d of the node:

$$b = \min\left(\lceil \frac{\log(1-\epsilon)}{d\log(1-v^\pi(\mathbf{s}))} \rceil, B\right), \tag{3}$$

where B denotes the upper bound of the budget and ϵ is the expected accuracy, thus a larger ϵ (e.g., 0.95) encourages more conservative searching. Besides, we empirically employ the $1/d$ term, which fosters exploration at the start of searching but encourages exploitation with d increasing to avoid search space explosion.

As the value scores of the preceding search steps usually suffer a larger variance due to the inefficient learning of delayed and sparse rewards (Sutton and Barto 2018), estimation of them is relatively not accurate enough. This inevitably influences the computation of suitable budget sizes. Therefore, we further propose to calibrate the value scores using the value of the corresponding trajectory from greedy decoding (denoted as $\hat{v}$), especially for the first few steps:

$$v'(\mathbf{s}) = \frac{v^\pi(\mathbf{s}) + \hat{v}/d}{1 + 1/d}, \tag{4}$$

where $v'(\mathbf{s})$ represents the calibrated value after normalization. We add the $1/d$ term to mainly adjust the value scores of the first several steps.

Expansion Strategies We propose two expansion strategies that prioritize efficiency and performance, respectively.

- *Incremental Expansion*: This strategy incrementally expands one child node after another. If the budget allows, the same node can be reselected until the budget is fully utilized. This method tends to conserve computational resources by carefully managing the budget.
- *Batch Expansion*: In contrast, this strategy consumes the entire budget allocated to the selected node during each iteration, resulting in the generation of multiple child nodes simultaneously. This method broadens the search space for subsequent iterations, potentially leading to the identification of superior nodes and enhancing overall performance.

[2] Specially, we define $\mathbf{s}_0$ as q.

3.2 Value Network

The value network $v^\pi(\mathbf{s})$ seeks to approximate the expected cumulative reward starting from state $\mathbf{s}$ and following a policy π thereafter. This can be represented as $v^\pi(\mathbf{s}) = \mathbb{E}_\pi [r_t \mid \mathbf{s}_t = \mathbf{s}]$, where r_t is the discounted return starting from state $\mathbf{s}_t$.

Particularly, given a question q and its correct answer a from an expert demonstration dataset. Each trajectory with reasoning steps (e.g., s_i) and final predicted answer $\hat{a}$ is firstly sampled from the policy π:

$$s_1, ..., s_n, \hat{a} \sim \pi(\mathcal{D}, q). \tag{5}$$

Then, we only take the answer correctness as distant supervision for each reasoning step to train the value network via a Mean Squared Error (MSE) loss:

$$\mathcal{L} = (v^\pi(\mathbf{s}_i) - \mathbb{I}[a = \hat{a}])^2, \tag{6}$$

where $\mathbb{I}(*)$ denotes an indicator function.

In this work, regardless of the policy used, we simply take Llama-3-8B[3] with a regressive head as our value network. This regressive head is a randomly initialized linear layer, which consumes the hidden state of the last input token and returns a scalar within $[0, 1]$:

$$v^\pi(\mathbf{s}_i) = \texttt{Head}(\texttt{Llama}(\mathbf{s}_i)[-1]). \tag{7}$$

4 Experiment

4.1 Setup

Dataset We conduct experiments on three popular mathematical reasoning datasets:

- **GSM8K** (Cobbe et al. 2021): This dataset comprises 7,473 training and 1,319 testing grade school math word problems that take $2 \sim 8$ steps to solve. Solutions primarily involve performing a sequence of elementary calculations using basic arithmetic operations.
- **TabMWP** (Lu et al. 2022): This dataset features 38,431 tabular math word problems, presented in either free-text or multiple-choice formats. We focus on the more general free-text category, consisting of 17,315 training questions and 1,000 randomly sampled test questions for evaluation.
- **MATH** (Hendrycks et al. 2021): It consists of 12,500 challenging competition mathematics problems. Following previous work (Lightman et al. 2023), we test on a subset of 500 cases, MATH500.

Models and Hyperparameters For GSM8K and TabMWP, we employ Mixtral-8×7B (Jiang et al. 2024) or Llama-3-8B as the policy model and train Llama-3-8B as the value network. Due to the difficulty of MATH, we adopt Llama-3-8B-Instruct, which has demonstrated exceptional mathematical abilities under system 1 mode after being fine-tuned on extensive post-training data. For the policy models, we adhere to the standard approach of utilizing 8 / 4 shots in-context learning for GSM8K / TabMWP, with a temperature of 0.6. We directly use the official 0-shot instruction for MATH testing, with a temperature of 0.8. By default, we set $N, B, \lambda, \varepsilon, \epsilon$ as 100, 10, 0, 0.8, and 0.9, respectively, and investigate other combinations in our analyses. For the value networks, we sample 8 trajectories per training instance, also with a temperature of 0.6. Then, we train the models for 1 epoch across all datasets, employing the AdamW optimizer (Loshchilov and Hutter 2017) with a learning rate of 5e-6 and a linear learning rate scheduler. Besides, we allocate 5% of the training instances as a development set to select the optimal checkpoints as value networks.

Evaluation Metrics We adopt answer *Accuracy* and the number of generated tokens (*#Tokens (k)*) as evaluation metrics for performance and cost, respectively. It should be noted that we do not take into account the cost of executing value networks following (Kang et al. 2024). This is because a value network only performs the regression task, which incurs significantly lower costs compared to the primary generation task. Besides, it can be deployed in parallel in practice.

Baselines We consider the following baselines:

- **Greedy Decoding**: It intuitively selects the most probable next token at each decoding step.
- **Hard Voting@K** (Wang et al. 2022): Known as self-consistency, which ensembles the answers from multiple sampled solutions as the final answer using majority voting. We sample $K = \{5, 10, 20\}$ times with a temperature of 0.6.
- **ToT-DFS** (Yao et al. 2024): We implement it by capitalizing on the guidance from our trained value network. Specifically, we prune a node if its value score falls below a threshold of 0.5 and limit the maximum number of children to 5 for preventing infinite loops.
- **ToT-BFS / BeamSearch** (Khalifa et al. 2023; Yao et al. 2024; Xie et al. 2024; Zhu et al. 2024): These two methods work similarly for this task. Again leveraging our value networks, each node is expanded to have 5 child nodes, and only 5 nodes with the highest value scores at each depth are kept to avoid search space explosion.
- **Soft Voting@K**: It is an enhancement over hard voting by utilizing our value networks. It softly ensembles the answers of different paths by taking their value scores as weights.

4.2 Main Results

Table 1 shows the main test results on GSM8K and TabMWP. We observe the following conclusions:

Value Guidance Boosts Model Performance In line with prior research (Wang et al. 2022), Hard Voting significantly improves Accuracy. However, its costs also proportionately increase with the growing of sampling size K. With the guidance of our value networks, both Soft Voting and tree search algorithms can further enhance Accuracy without incurring additional costs. Besides, *Soft Voting@5* consistently surpasses *Hard Voting@20*, substantiating the effectiveness of verification as previously discussed in (Cobbe et al. 2021).

[3]https://ai.meta.com/blog/meta-llama-3/

		GSM8K		TabMWP	
		Accuracy ↑	#Tokens (k) ↓	Accuracy ↑	#Tokens (k) ↓
Mixtral-8×7B	Greedy Decoding	.607	0.14	.762	0.07
	Hard Voting@5	.705	0.66	.761	0.37
	Hard Voting@10	.740	1.32	.782	0.73
	Hard Voting@20	.769	2.63	.796	1.46
	ToT-DFS	.722	**0.22**	.822	**0.16**
	ToT-BFS	.801	2.22	.861	1.45
	Soft Voting@5	.779	0.66	.811	0.37
	Soft Voting@10	.830	1.32	.832	0.73
	Soft Voting@20	**.843**	2.63	.847	1.46
	Ours (Incremental)	.797	0.41	**.863**	0.22
	Ours (Batch)	.823	0.55	.854	0.29
Llama3-8B	Greedy Decoding	.485	0.18	.659	0.08
	Hard Voting@5	.572	0.57	.680	0.42
	Hard Voting@20	.667	2.38	.698	1.68
	ToT-DFS	.676	**0.24**	.704	**0.19**
	ToT-BFS	.756	1.89	.787	1.35
	Soft Voting@5	.689	0.57	.747	0.42
	Soft Voting@20	**.770**	2.38	**.796**	1.68
	Ours (Incremental)	.731	0.46	.779	0.27
	Ours (Batch)	.757	0.59	.776	0.35

Table 1: Main test results, where *Ours (Incremental)* and *Ours (Batch)* denote LiteSearch equipped with Incremental and Batch Expansion strategies, respectively. For methods guided by our value networks, we emphasize the best results in **bold** and the second / third-best results with underlining.

	Accuracy ↑	#Tokens (k) ↓
Greedy Decoding	.280	0.32
Hard Voting@5	.312	1.56
Hard Voting@10	.338	3.12
Soft Voting@5	.378	1.56
Soft Voting@10	.392	3.12
Ours (Incremental)	**.420**	2.36
Ours (Batch)	.412	2.43

Table 2: Results on MATH500 using Llama-3-8B-Instruct.

	Accuracy ↑	#Tokens (k) ↓
Ours (Incremental)	**.797**	0.41
⇒ static budget	.779	0.67
w/o depth penalty	.780	0.43
w/o greedy value	.783	**0.40**
Ours (Batch)	**.823**	**0.55**
⇒ static budget	.802	1.79
w/o depth penalty	.815	0.79
w/o greedy value	.806	0.62

Table 3: Ablation study on dynamic budgeting.

Current Tree Search Algorithms Neglect the Performance-Cost Tradeoff Previous methods, *ToT-DFS* and *ToT-BFS*, prefer different evaluation metrics. Among the value-guided approaches, *ToT-DFS* consistently has the lowest cost but achieves suboptimal performance. This is because *ToT-DFS* focuses mainly on pruning bad nodes and lacks the flexibility to select better nodes for further improvement. In contrast, *ToT-BFS* tackles this shortcoming of *ToT-DFS* by maintaining a branch of nodes with the highest values, thereby resulting in better performance. However, it also unnecessarily visits lots of nodes during the search, leading to significantly higher costs.

LiteSearch Maintains Performance and Decreases Cost By fully utilizing the guidance from value networks, our methods achieve the best tradeoff between performance and cost. Our approaches fall within the cost range of *ToT-DFS* and *Soft Voting@5*, yet yield significantly better performance. For the two expansion strategies, *Ours (Incremental)* saves nearly 20% of costs of *Ours (Batch)* and performs even better on TabMWP. However, *Ours (Incremental)* performs noticeably worse than *Ours (Batch)* on Accuracy on GSM8K, with a 2.6-point lower score. This is due to the Batch Expansion strategy providing a better comparison among nodes for selection by expanding more nodes each time. We extensively experiment on another more challenging MATH500 using Llama-3-8B-Instruct. The results in Table 2 again demonstrate the effectiveness of our methods.

4.3 Ablation Study and Analyses

Dynamic Budgeting Helps Both Performance and Cost-Efficiency We first study the effectiveness of the dynamic budget size b, which is decided by Eq. 3. The following variants are considered: (1) ⇒ *static budget*: We directly set b as B, resulting in each node being expanded with a fixed

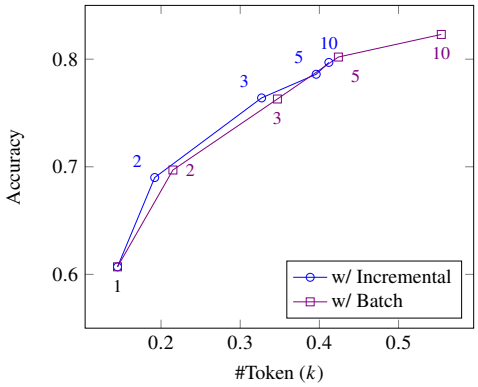

Figure 2: Performance of *Ours (Incremental)* and *Ours (Batch)* on GSM8K when using different budget upperbounds B, where $B = \{1, 2, 3, 5, 10\}$.

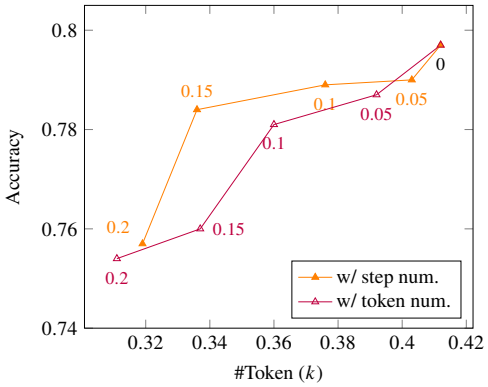

Figure 3: Performance of *Ours (Incremental)* on GSM8K when using *step number* and *token number* to estimate progress term $p(\mathbf{s})$, where $\lambda = \{0, 0.05, 0.1, 0.15, 0.2\}$.

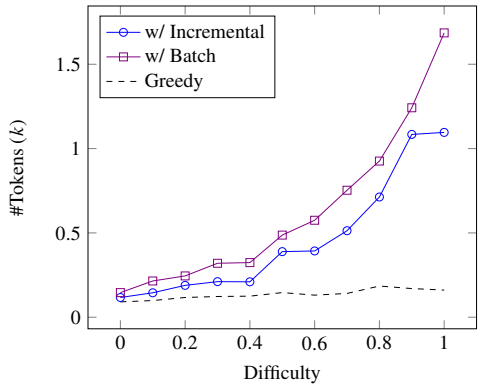

Figure 4: Cost of *Ours (Incremental)* and *Ours (Batch)* with the growth of difficulty on GSM8K. *Difficulty* is defined as "$1 - x$", where x is the frequency of gold answer in 20 sampled paths.

budget size; (2) *w/o depth penalty*: We remove the $1/d$ term from Eq. 3, which previously penalized b as the depth d increased; (3) *w/o greedy value*: We do not consider Eq. 4 to calibrate value scores with greedy results.

As shown in Table 3, we observe that dynamic budgeting helps in both *Ours (Incremental)* and *Ours (Batch)* by allowing them to maintain higher accuracy with fewer tokens compared to all other variants. Specifically, $\Rightarrow$ *static budget* severely hurts both performance and cost, particularly leading to 3 times computation costs when using Batch Expansion. *w/o depth penalty* and *w/o greedy value* perform competitively for *Ours (Incremental)*, but still have considerable negative influence on *Ours (Batch)*. These results highlight the importance of dynamic budgeting especially in scenarios where Batch Expansion is employed.

Influences of Budget Limitation Budget limitation B decides the upperbound of budget size b. As illustrated in Fig. 2, we observe a clear tradeoff between performance and cost. With the growth of B, the computation cost also increases correspondingly because larger budget sizes are allocated to challenging states with lower value scores. Consequently, more problems are correctly solved due to more comprehensive searching. Regarding the two expansion strategies, *Ours (Incremental)* perform slightly better than *Ours (Batch)* with competitive accuracy but fewer costs when $B \leq 3$. This is because it may not use up all budgets when good nodes have been generated during incremental expansion. However, *Ours (Batch)* yields better accuracy by taking more costs when $B = \{5, 10\}$ because it fully utilizes allocated budgets, thus providing larger search space for better selection.

Influence of Progress Estimation We then investigate the choice of $p(\mathbf{s})$ and λ in Eq. 2. We consider *step number* and *token number* against corresponding results of greedy decoding to estimate $p(\mathbf{s})$. As depicted in Fig. 3, increasing λ improves cost-efficiency by prioritizing nodes with faster progress at the risk of inaccuracy. Comparing *step number* and *token number*, the former is relatively better with a modest downward trend. By sacrificing 1.3 points in accuracy, utilizing *step number* and $\lambda = 0.15$ saves nearly 20% computational costs. In contrast, the efficacy of *token number* is unsatisfactory. This can be attributed to its higher degree of variability, thus yielding less precise estimates of progress terms.

Harder Problems are Allocated Larger Budgets Fig. 4 illustrates the correlation between cost and question difficulty. Inspired by (Wang et al. 2024b), we estimate the difficulty of a question by computing the frequency of the gold answer in multiple sampled paths after inversion ("$1 - x$"). We observe that for easier questions, our methods cost competitively to *Greedy Decoding*. However, as the difficulty escalates, the cost of our methods also rises proportionately. Regarding our expansion strategies, *Ours (Batch)* consistently takes higher costs and the gap also widens with the difficulty increases.

Mixture-of-Domain Boosts Performance An important future direction is to construct a general value network that can address questions from different domains. To validate the potential of this direction, we conduct experiments us-

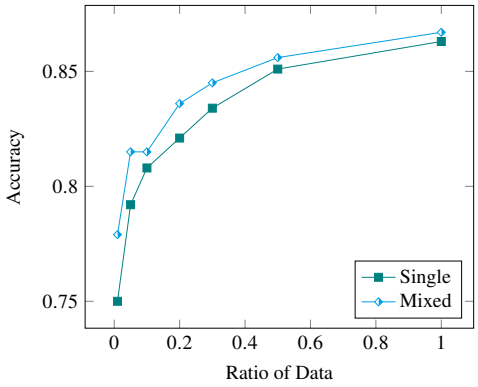

Figure 5: Accuracy of *Ours (Incremental)* on TabMWP with value networks trained with *Single*- or *Mixed*-domain data, where we use full GSM8K and different ratios of TabMWP from 1% to 100%.

Question:
Josh decides to try flipping a house. He buys a house for $80,000 and then puts in $50,000 in repairs. This increased the value of the house by 150%. How much profit did he make?

Correct Solution:
Step 1. The value of the house increased by 150%, so it increased by 150% * 80,000 = 120,000. **(*Value: 0.33*)**
Step 2. Thus, the value of the house is now 80,000 + 120,000 = 200,000. **(*Value: 0.49*)**
Step 3. Josh put in 50,000 + 80,000 = 130,000 in the house, so he made a profit of 200,000 - 130,000 = 70,000. **(*Value: 0.97*)**

Wrong Solution:
Step 1. The total cost was 80,000 + 50,000 = 130,000 **(*Value: 0.62*)**
Step 2. The new value of the house was 150% of the cost or 1.5 * 130,000 = 195,000 **(*Value: 0.31*)** ✗
Step 3. So the profit was 195,000 - 130,000 = 65,000 **(*Value: 0.35*)** ✗

Figure 6: A lemon picked example. We provide correct and wrong solutions with value scores for each step and error steps are marked.

ing value networks trained with different ratios of TabMWP data and full GSM8K data. Despite the significant difference in question style and answer type, the results in Fig. 5 demonstrate that using a mixture of different domains helps improve search performance, especially when the target-domain training instances are scarce (0.75 vs. 0.78 on Accuracy when using 1% TabMWP data). This highlights the effectiveness of building robust and stronger value networks by collecting various training instances. Further exploration in this direction will be pursued in future work.

Voting Helps when Values are Inaccurate Analyses above have shown the effectiveness of our method. However, though much more efficient, guided tree searches often cannot outperform Soft Voting on Accuracy. We first collect the questions that our approach fails to solve, yet are successfully addressed by *Soft Voting@20*. Fig. 6 displays a lemon pick example. We observe that our value network can select the correct answer when the complete rationale

	Accuracy ↑	#Tokens (k) ↓
Best@20	.833	2.63
Soft Voting@20	.843	2.63
Ours (Batch)	.823	**0.55**
+Soft Voting (α = 0.7)	.841	0.73
+Soft Voting (α = 0.8)	.843	0.79
+Soft Voting (α = 0.9)	**.847**	0.99

Table 4: Results of our methods enhanced by voting, where *Best@20* is a variant of *Soft Voting@20*, which only selects the best path with the highest value as the final output. For *Ours + Soft Voting*, we discard results with values lower than α, and utilize *Soft Voting@20* to solve them.

is provided. However, the first two steps in the correct path are scored much lower than the first step of the wrong solution. This results in a reduced priority in exploring the node that is actually of higher quality. Besides, due to the imperfect value network, some incorrect paths may be erroneously scored higher than the correct ones. Guided tree search methods, which search for only one path as the final answer, inevitably fail in these instances. However, Soft Voting can mitigate this issue by leveraging the benefits of both majority voting and the value network. Consequently, even if the highest value is attained by an incorrect path, Soft Voting still has the potential to reach the correct answer with a higher frequency. As demonstrated in Table 4, the use of voting enables *Soft Voting@20* to outperform *Best@20*, highlighting the efficacy of voting in enhancing accuracy.

Inspired by these findings, we further investigate the improvement of our method using voting. Specifically, we discard the answers predicted by our method when their value scores fall below a threshold α. Generally, these predictions exhibit a higher error rate due to the correlation between value scores and correctness. Subsequently, we employ Soft Voting to address these unresolved questions. The results in Table 4 indicate that accuracy can be significantly improved by increasing α. However, the associated costs also rise substantially, albeit remaining lower than those of *Soft Voting@20*.

5 Conclusion

In this work, we study guided tree search to address math problems, aiming to decrease the computation costs while maintaining the performance. Inspired by the theory of value function, we propose dynamic node selection and expansion strategies, which dynamically determine the priority of nodes to explore and manage the computational budget during expansion. Both procedures are guided by an easy-to-implement value network trained without step-wise supervision. Experiments show that our methods achieve competitive performance with typical baselines but significantly save computation costs. Ablation studies validate the effectiveness of each component, providing more feasible options for various practical scenarios. Besides, we identify the shortcomings of this research line, and provide a potential strategy for addressing these issues.

Acknowledgments

The project was supported by National Key R&D Program of China (No.2022ZD0160501), Natural Science Foundation of Fujian Province of China (No.2024J011001), and the Public Technology Service Platform Project of Xiamen (No.3502Z20231043). We also thank the reviewers for their insightful comments.

References

Achiam, J.; Adler, S.; Agarwal, S.; Ahmad, L.; Akkaya, I.; Aleman, F. L.; Almeida, D.; Altenschmidt, J.; Altman, S.; Anadkat, S.; et al. 2023. Gpt-4 technical report. *arXiv:2303.08774*.

Amini, A.; Gabriel, S.; Lin, S.; Koncel-Kedziorski, R.; Choi, Y.; and Hajishirzi, H. 2019. MathQA: Towards Interpretable Math Word Problem Solving with Operation-Based Formalisms. In *Proc. of NAACL 2019*, 2357–2367.

Chen, Z.; Deng, Y.; Yuan, H.; Ji, K.; and Gu, Q. 2024. Self-play fine-tuning converts weak language models to strong language models. *arXiv:2401.01335*.

Cobbe, K.; Kosaraju, V.; Bavarian, M.; Chen, M.; Jun, H.; Kaiser, L.; Plappert, M.; Tworek, J.; Hilton, J.; Nakano, R.; et al. 2021. Training verifiers to solve math word problems. *arXiv:2110.14168*.

Daniel, K. 2017. *Thinking, fast and slow*.

Feng, X.; Wan, Z.; Wen, M.; Wen, Y.; Zhang, W.; and Wang, J. 2023. Alphazero-like tree-search can guide large language model decoding and training. *arXiv:2309.17179*.

Hendrycks, D.; Burns, C.; Kadavath, S.; Arora, A.; Basart, S.; Tang, E.; Song, D.; and Steinhardt, J. 2021. Measuring Mathematical Problem Solving With the MATH Dataset. In *Proc. of NIPS 2021*.

Hopkins, M.; Petrescu-Prahova, C.; Levin, R.; Le Bras, R.; Herrasti, A.; and Joshi, V. 2017. Beyond sentential semantic parsing: Tackling the math SAT with a cascade of tree transducers. In *Proc. of EMNLP 2017*, 795–804.

Huang, J.; Cui, L.; Wang, A.; Yang, C.; Liao, X.; Song, L.; Yao, J.; and Su, J. 2024. Mitigating Catastrophic Forgetting in Large Language Models with Self-Synthesized Rehearsal. In *Proc. of ACL 2024*, 1416–1428.

Jiang, A. Q.; Sablayrolles, A.; Roux, A.; Mensch, A.; Savary, B.; Bamford, C.; Chaplot, D. S.; Casas, D. d. l.; Hanna, E. B.; Bressand, F.; et al. 2024. Mixtral of experts. *arXiv:2401.04088*.

Kang, J.; Li, X. Z.; Chen, X.; Kazemi, A.; and Chen, B. 2024. MindStar: Enhancing Math Reasoning in Pre-trained LLMs at Inference Time. *arXiv:2405.16265*.

Khalifa, M.; Logeswaran, L.; Lee, M.; Lee, H.; and Wang, L. 2023. Grace: Discriminator-guided chain-of-thought reasoning. In *Proc. of Findings of EMNLP 2023*, 15299–15328.

Kojima, T.; Gu, S. S.; Reid, M.; Matsuo, Y.; and Iwasawa, Y. 2022. Large language models are zero-shot reasoners. In *Proc. of NIPS 2022*, 22199–22213.

Li, J.; Wang, L.; Zhang, J.; Wang, Y.; Dai, B. T.; and Zhang, D. 2019. Modeling intra-relation in math word problems with different functional multi-head attentions. In *Proc. of ACL 2019*, 6162–6167.

Lightman, H.; Kosaraju, V.; Burda, Y.; Edwards, H.; Baker, B.; Lee, T.; Leike, J.; Schulman, J.; Sutskever, I.; and Cobbe, K. 2023. Let's Verify Step by Step. *arXiv:2305.20050*.

Liu, B.; Lyu, C.; Min, Z.; Wang, Z.; Su, J.; and Wang, L. 2023. Retrieval-augmented multi-modal chain-of-thoughts reasoning for large language models. *arXiv:2312.01714*.

Liu, H.; and Yao, A. C.-C. 2024. Augmenting math word problems via iterative question composing. *arXiv:2401.09003*.

Loshchilov, I.; and Hutter, F. 2017. Decoupled weight decay regularization. *arXiv:1711.05101*.

Lu, P.; Qiu, L.; Chang, K.-W.; Wu, Y. N.; Zhu, S.-C.; Rajpurohit, T.; Clark, P.; and Kalyan, A. 2022. Dynamic Prompt Learning via Policy Gradient for Semi-structured Mathematical Reasoning. In *Proc. of ICLR 2022*.

Luo, H.; Sun, Q.; Xu, C.; Zhao, P.; Lou, J.; Tao, C.; Geng, X.; Lin, Q.; Chen, S.; and Zhang, D. 2023. Wizardmath: Empowering mathematical reasoning for large language models via reinforced evol-instruct. *arXiv:2308.09583*.

Ma, Q.; Zhou, H.; Liu, T.; Yuan, J.; Liu, P.; You, Y.; and Yang, H. 2023. Let's reward step by step: Step-Level reward model as the Navigators for Reasoning. *arXiv:2310.10080*.

Madaan, A.; Tandon, N.; Gupta, P.; Hallinan, S.; Gao, L.; Wiegreffe, S.; Alon, U.; Dziri, N.; Prabhumoye, S.; Yang, Y.; et al. 2024. Self-refine: Iterative refinement with self-feedback. In *Proc. of NIPS 2024*.

Matsuzaki, T.; Ito, T.; Iwane, H.; Anai, H.; and Arai, N. H. 2017. Semantic parsing of pre-university math problems. In *Proc. of ACL 2017*, 2131–2141.

Qin, J.; Liang, X.; Hong, Y.; Tang, J.; and Lin, L. 2021. Neural-Symbolic Solver for Math Word Problems with Auxiliary Tasks. In *Proc. of ACL 2021*, 5870–5881.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement learning: An introduction*. MIT press.

Suzgun, M.; and Kalai, A. T. 2024. Meta-prompting: Enhancing language models with task-agnostic scaffolding. *arXiv:2401.12954*.

Tian, Y.; Peng, B.; Song, L.; Jin, L.; Yu, D.; Mi, H.; and Yu, D. 2024. Toward Self-Improvement of LLMs via Imagination, Searching, and Criticizing. *arXiv:2404.12253*.

Touvron, H.; Martin, L.; Stone, K.; Albert, P.; Almahairi, A.; Babaei, Y.; Bashlykov, N.; Batra, S.; Bhargava, P.; Bhosale, S.; et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv:2307.09288*.

Wang, A.; Song, L.; Peng, B.; Jin, L.; Tian, Y.; Mi, H.; Su, J.; and Yu, D. 2024a. Improving LLM Generations via Fine-Grained Self-Endorsement. In *Proc. of ACL 2024 Findings*, 8424–8436.

Wang, A.; Song, L.; Tian, Y.; Peng, B.; Jin, L.; Mi, H.; Su, J.; and Yu, D. 2024b. Self-Consistency Boosts Calibration for Math Reasoning. *arXiv:2403.09849*.

Wang, C.; Deng, Y.; Lv, Z.; Yan, S.; and Bo, A. 2024c. Q*: Improving Multi-step Reasoning for LLMs with Deliberative Planning. *arXiv:2406.14283*.

Wang, P.; Li, L.; Shao, Z.; Xu, R.; Dai, D.; Li, Y.; Chen, D.; Wu, Y.; and Sui, Z. 2023. Math-shepherd: A label-free step-by-step verifier for llms in mathematical reasoning. *arXiv:2312.08935*.

Wang, X.; Wei, J.; Schuurmans, D.; Le, Q. V.; Chi, E. H.; Narang, S.; Chowdhery, A.; and Zhou, D. 2022. Self-Consistency Improves Chain of Thought Reasoning in Language Models. In *Proc. of ICLR 2022*.

Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Xia, F.; Chi, E.; Le, Q. V.; Zhou, D.; et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Proc. of NIPS 2022*, 24824–24837.

Xie, Y.; Kawaguchi, K.; Zhao, Y.; Zhao, J. X.; Kan, M.-Y.; He, J.; and Xie, M. 2024. Self-evaluation guided beam search for reasoning. In *Proc. of NIPS 2024*.

Yang, C.; Wang, X.; Lu, Y.; Liu, H.; Le, Q. V.; Zhou, D.; and Chen, X. 2024. Large Language Models as Optimizers. In *Proc. of ICLR 2024*.

Yao, S.; Yu, D.; Zhao, J.; Shafran, I.; Griffiths, T.; Cao, Y.; and Narasimhan, K. 2024. Tree of thoughts: Deliberate problem solving with large language models. In *Proc. of NIPS 2024*.

Yu, L.; Jiang, W.; Shi, H.; Jincheng, Y.; Liu, Z.; Zhang, Y.; Kwok, J.; Li, Z.; Weller, A.; and Liu, W. 2023. MetaMath: Bootstrap Your Own Mathematical Questions for Large Language Models. In *Proc. of ICLR 2023*.

Zeng, Z.; Watson, W.; Cho, N.; Rahimi, S.; Reynolds, S.; Balch, T.; and Veloso, M. 2023. FlowMind: automatic workflow generation with LLMs. In *Proc. of ACM ICAIF 2023*, 73–81.

Zhang, D.; Li, J.; Huang, X.; Zhou, D.; Li, Y.; and Ouyang, W. 2024. Accessing GPT-4 level Mathematical Olympiad Solutions via Monte Carlo Tree Self-refine with LLaMa-3 8B. *arXiv:2406.07394*.

Zhou, P.; Pujara, J.; Ren, X.; Chen, X.; Cheng, H.-T.; Le, Q. V.; Chi, E. H.; Zhou, D.; Mishra, S.; and Zheng, H. S. 2024. Self-discover: Large language models self-compose reasoning structures. *arXiv:2402.03620*.

Zhu, T.; Zhang, K.; Xie, J.; and Su, Y. 2024. Deductive Beam Search: Decoding Deducible Rationale for Chain-of-Thought Reasoning. *arXiv:2401.17686*.