# Ensembling Large Language Models with Process Reward-Guided Tree Search for Better Complex Reasoning

**Anonymous ACL submission**


## Abstract

Despite recent advances in large language models, open-source models often struggle to consistently perform well on complex reasoning tasks. Existing ensemble methods, whether applied at the token or output levels, fail to address these challenges. In response, we present Language model Ensemble with Monte Carlo Tree Search (LE-MCTS), a novel framework for process-level ensembling of language models. LE-MCTS formulates step-by-step reasoning with an ensemble of language models as a Markov decision process. In this framework, states represent intermediate reasoning paths, while actions consist of generating the next reasoning step using one of the language models selected from a predefined pool. Guided by a process-based reward model, LE-MCTS performs a tree search over the reasoning steps generated by different language models, identifying the most accurate reasoning chain. Experimental results on five mathematical reasoning benchmarks demonstrate that our approach outperforms both single language model decoding algorithms and language model ensemble methods. Notably, LE-MCTS improves performance by 3.6% and 4.3% on the MATH and MQA datasets, respectively, highlighting its effectiveness in solving complex reasoning problems.


## 1 Introduction

Large language models (LLMs) have demonstrated superior performance across a range of tasks, primarily due to their large capacity and high-quality training data. However, unlike prominent closed-source LLMs such as GPT-4 (OpenAI, 2024) and Gemini-1.5 (Gemini, 2024), open-source models like Mistral (Jiang et al., 2023a), LLaMA-3 (LLaMA, 2024), and Gemma-2 (Gemma, 2024) are constrained by factors such as data availability, architecture, and hyperparameter choices. As a result, they exhibit different strengths and weaknesses (Jiang et al., 2023b).

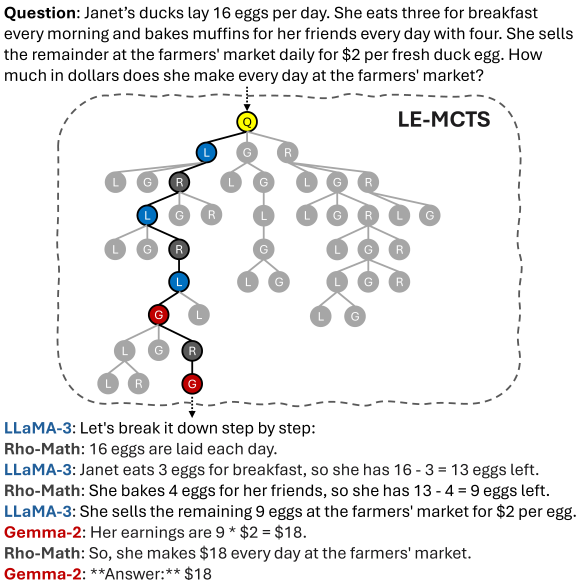


Figure 1: **Example output of LE-MCTS.** The reasoning steps in the LE-MCTS output can be generated by different LLMs. We highlight the root node in yellow and apply the same color coding to the corresponding nodes and the language model.

Language model ensembling is a well-known approach for creating a versatile model by combining weaker language models. Previous studies on LM ensembling have focused on merging language models at the token and output levels. Token-level approaches have merged the output logits or probabilities of language models, using perplexity-based weighted averaging (Liu et al., 2024b; Mavromatis et al., 2024) and vocabulary projection (Xu et al., 2024). Although token-level ensemble methods offer fine-grained fusion of language models, they face several constraints related to tokenizer vocabularies and model architectures, requiring the training of additional projection matrices (Xu et al., 2024) to mitigate these. Output-level approaches have involved ensembling fully generated outputs, either by ranking multiple outputs to select the best one (Farinhas et al., 2023;

Jiang et al., 2023b) or by using an additional fusion model to fuse them (Jiang et al., 2023b). Although output-level ensembling can be applied to any language model, it cannot produce a correct answer if all candidate outputs are flawed (Xu et al., 2024).

Our experiments in Table 2 show that LLMs have varying levels of expertise across different types of math problems. Such disparities cannot be easily mitigated by applying decoding algorithms designed for a single LLM. Furthermore, the results show that both the existing token-level and output-level LM ensemble methods perform worse than a single language model. These results clearly demonstrate that current LM ensemble methods are particularly ineffective in solving complex reasoning problems, pointing to the need for a framework specifically designed to handle such tasks and to ensure consistently high performance across diverse reasoning problems.

To address the limitations of token- and output-level ensembling, we propose a process-level language model ensemble framework tailored to complex reasoning tasks. Many complex reasoning problems can be solved step by step, and evaluating each reasoning step individually allows the model to correct errors early, thus guiding the decoding process toward more accurate solutions (Yao et al., 2024; Besta et al., 2024; Yu et al., 2024; Ma et al., 2023). This approach is further supported by recent advances in process-based reward models (PRMs) (Lightman et al., 2023; Wang et al., 2024b; Lu et al., 2024a), which make it possible to measure the correctness of each intermediate reasoning step as a scalar value. Based on the advantages of process-level reasoning, process-level ensembling offers finer and more efficient control over generation compared to output-level ensembling, as it allows for the correction of intermediate reasoning steps without regenerating entire solutions. Additionally, process-level ensembling is less constrained than token-level ensemble approaches, as it eliminates the need to match vocabulary and architecture.

We present Language model Ensemble with Monte Carlo Tree Search (LE-MCTS), a pioneering framework for process-level ensembling of language models. We formulate the step-by-step reasoning involving multiple LLMs as a Markov decision process (MDP) (Bellman, 1957). Specifically, the state is defined as the intermediate reasoning steps, and the action is defined as generating the next reasoning step using one of the language models selected from a predefined pool of LLMs. Our MCTS algorithm, inspired by AlphaZero (Silver et al., 2017), performs a tree search over the unified space of reasoning steps generated by different LLMs. By following the guidance from PRM, we can obtain the reasoning chain that is likely to be the most accurate among the possible combinations of reasoning steps generated by the LLMs.

We evaluate LE-MCTS on five math reasoning benchmarks: GSM8K (Cobbe et al., 2021), MATH (Hendrycks et al., 2021), SVAMP (Patel et al., 2021), ASDiv (Miao et al., 2020), and MQA (Amini et al., 2019). Our results show that LE-MCTS consistently outperforms or matches existing LM ensemble approaches across all tasks. We also evaluate a straightforward integration of process reward-guided decoding algorithms with the LM ensemble. Although these models perform well on grade school math problems, they fail on more complex datasets, such as MATH and MQA. In contrast, LE-MCTS improves performance by 3.6% and 4.3% on MATH and MQA, respectively, compared to the second-best models. These experimental results highlight the effectiveness of LE-MCTS in tackling complex reasoning problems.

## 2 Methods

First, we introduce the problem formulation and key notation. We then describe our LE-MCTS algorithm.

### 2.1 Problem Setup

Given an input problem $q$ and $L$ language models $\{\pi_1, \ldots, \pi_L\}$, a language model $\pi_l$ can generate a complete output $o$ or a reasoning step $p_k \sim \pi_l(\cdot \mid q, p_{1:k-1})$, where $p_{1:k-1}$ is the sequence of previously generated reasoning steps up to $p_k$. We define each sentence ending with a newline character as a reasoning step. This setup allows us to compare reasoning steps from different language models, $\{p_k^1, \ldots, p_k^L\}$, which are generated from the same intermediate reasoning trajectory $p_{1:k-1}$. Our objective is to find the optimal combination of reasoning steps generated by different LMs, denoted as $o^* = p_{1:K}^*$, where $p_K$ is the final reasoning step that includes the answer. Ideally, we can obtain $o^*$ by evaluating all elements in the set $P = \prod_{k=1}^{K} P_k$, where $P_k = \{p_k^1, \ldots, p_k^L\}$. While $o^*$ offers a better solution compared to a single LM output $o^l$, it becomes infeasible to evaluate the entire $P$ as $L$ and $K$ increase.

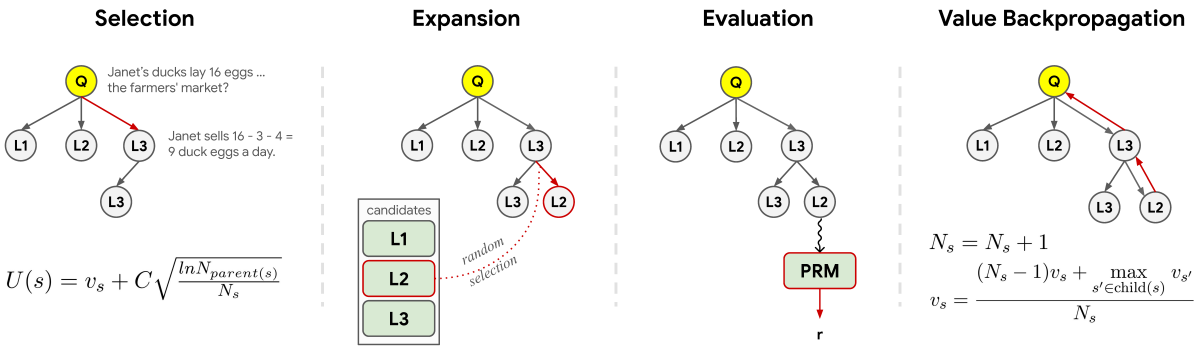


Figure 2: **Single iteration of LE-MCTS.** This example illustrates an ensemble of three LLMs. The iteration is repeated until the maximum number of iterations, $n_{iter}$, is reached or no further nodes in the tree can be expanded.

To address this issue, we model the step-by-step reasoning problem as a Markov decision process (MDP) (Bellman, 1957) and solve it using Monte Carlo Tree Search (MCTS) (Coulom, 2006). Specifically, the root node represents the input $q$, while each child node $s$ corresponds to an intermediate reasoning trajectory $p_{1:k}$, storing its value $v_s$ and visit count $N_s$. The value $v_s$ represents the accuracy of the intermediate reasoning trajectory. An action is defined as generating $p_k^l$ from $p_{1:k-1}$ using $\pi_l$, selected from $\{\pi_1, ..., \pi_L\}$. PRM computes the reward to guide MCTS towards maximizing the process reward for both intermediate reasoning steps and the final output. In the following sections, we introduce how LE-MCTS effectively integrates reasoning steps from various LMs into the tree and performs a lookahead search in this setup.

## 2.2 LE-MCTS

Each iteration of our LE-MCTS algorithm consists of four stages: selection, expansion, evaluation, and value backpropagation. LE-MCTS repeats the iterations until the number of iterations reaches the maximum value $n_{iter}$, or no further nodes in the tree can be expanded. A pseudocode for LE-MCTS is provided in Table 1.

**Selection** LE-MCTS begins each iteration with the selection phase, starting at the root node and hierarchically selecting child nodes until a leaf node $s_{\text{leaf}}$ is reached. To incorporate the quality values of nodes, we use the UCT algorithm (Kocsis and Szepesvári, 2006) for selecting a child node:

$$U(s) = v_s + C\sqrt{\frac{\ln N_{\text{parent}(s)}}{N_s}} \quad (1)$$

where $N_{\text{parent}(s)}$ is the visit count of $s$'s parent node, and $C$ is a constant controlling the exploration-exploitation trade-off. At each intermediate node, we select the child with the highest $U(s)$. A higher $C$ promotes exploration of underexplored nodes, while a lower $C$ favors searching for high-value nodes first. Terminal state nodes are excluded from selection, as the goal of LE-MCTS selection is to identify the best incomplete intermediate reasoning trajectory. The selection enables LE-MCTS to explore high-accuracy trajectories effectively without exhaustively examining the entire set $P$.

**Expansion** After selecting the leaf node $s_{\text{leaf}}$, we add a new child node to it. For simplicity, we assume $s_{\text{leaf}}$ corresponds to the intermediate reasoning trajectory $p_{1:k-1}$ in the following sections.

First, we randomly select a language model $\pi_l$ from the pool $\{\pi_1, \ldots, \pi_L\}$. We then greedily decode the next reasoning step $p_k$ using $\pi_l$ until a newline character appears:

$$p_{k,t} = \arg\max_{w \in V} \pi_l(w \mid p_{k,<t}; q, p_{1:k-1}) \quad (2)$$

where $V$ is the vocabulary. The expansion phase plays a key role in the process-level ensemble because it introduces new reasoning steps generated by different language models into the tree. We do not expand fully-expanded nodes that meet any of the following criteria:

- The number of child nodes reaches the constant $n_{\text{child}}$
- $v_s - \max_{s' \in \text{child}(s)} v_{s'} < \varepsilon$

The second criterion encourages LE-MCTS to explore deeper reasoning paths by prioritizing depth over breadth in its expansions. In complex reasoning problems, the length of the reasoning chain tends to increase with difficulty of the problem. Therefore, this criterion enables LE-MCTS to explore sufficiently deep reasoning trajectories within a limited number of MCTS iterations.

**Evaluation** In the evaluation phase, we compute the reward of the reasoning step $p_k$. Following the approach used in AlphaZero and its variants (Silver et al., 2017; Feng et al., 2023), we do not perform any rollouts. Instead, we directly employ the process-based reward model (PRM) to compute the reward. Specifically, we utilize a pre-trained PRM $\phi$ from Math Shepherd (Wang et al., 2024b), which takes the reasoning step $p_k$ and the problem $q$ as inputs, predicting the process reward $r_k = \phi(q, p_k)$.

One advantage of performing rollouts in our setup is that it enables the use of an outcome-based reward model (ORM) to compute the reward. However, as demonstrated by Uesato et al. (2022), both PRM and ORM emulate process-based feedback and achieve similar performance. Therefore, we decide not to perform rollouts, as the performance improvement with ORM is minimal, while the execution time increases by approximately five to tenfold.

**Value Backpropagation** After expanding and evaluating a leaf node, the statistics are propagated up the tree, and each node visited during the selection phase updates its value and visit count. The standard practice (Browne et al., 2012) for value backpropagation is increasing the visit count $N_s$ by 1 and then updating the node's value according to the following equation:

$$v_s = \frac{(N_s - 1)v_s + r_k}{N_s} \tag{3}$$

With the standard backpropagation strategy, the reasoning steps generated by different language models contribute equally to $v_s$. However, in our setup, it is sufficient to identify at least one intermediate reasoning trajectory where any language model can generate a viable subsequent reasoning path. Thus, a node is considered acceptable as long as it has at least one child with a high value, even if the others have low values.

To address this, we propose a new backpropagation strategy, *optimistic backpropagation*, which updates the node's value based on the maximum value among its child nodes instead of the reward:

$$v_s = \frac{(N_s - 1)v_s + \max\limits_{s' \in \text{child}(s)} v_{s'}}{N_s} \tag{4}$$

This optimistic backpropagation strategy disregards the signals from low-valued sibling nodes and directs LE-MCTS to focus on the most promising reasoning path.

**Algorithm: LE-MCTS**

**Input**: input $q$, language models $\{\pi_1, \ldots, \pi_L\}$, max MCTS iterations $n_{iter}$, UCT constant $C$, max # child nodes $n_{child}$, threshold $\varepsilon$, PRM $\phi$

```
// Initialize
s_0 ← CreateNode(T, q)
for i = 1, ..., n_iter do
  s ← s_0
  // Selection
  while s is not a leaf node do
    S ← {}
    for s' ∈ child(s) do
      if n(child(s')) < n_child and v'_s − max_{s''∈child(s')} v_{s''} ≥ ε then
        S ← S + {s'}
      end if
    end for
    s ← arg max_{s'∈S} v_{s'} + C sqrt(ln N_s / N_{s'})
  end while
  // Expansion
  π_l ← RandomSelect({π_1, ..., π_L})
  p_{1:k−1} ← GetPath(s)
  while p_{k,t} is not \n do
    p_{k,t} ← arg max_{w∈V} π_l(w | p_{k,<t}; q, p_{1:k−1})
  end while
  s' ← CreateNode(T, {p_{1:k−1}, p_k})
  // Evaluation and Value Backpropagation
  v_{s'} ← φ(q, p_k), N_{s'} ← 1
  while s' is not a root node do
    s' ← GetParent(s')
    N_{s'} ← N_{s'} + 1
    v_{s'} ← ((N_{s'} − 1)v_{s'} + max_{s''∈child(s')} v_{s''}) / N_{s'}
  end while
end for
Return ChooseBest(T)
```

**Output**: Highest-rewarded solution $p^*_{1:K}$

Table 1: Pseudocode for LE-MCTS.

# 3 Experiments

## 3.1 Experimental Settings

**Datasets and Evaluation** We conducted our experiments using five widely used math reasoning datasets: GSM8K (Cobbe et al., 2021), MATH (Hendrycks et al., 2021), SVAMP (Patel et al., 2021), ASDiv (Miao et al., 2020), and MQA (Amini et al., 2019). For the MATH dataset, we used the MATH500 subset to avoid data leakage, which is identical to the test set used by Lightman et al. (2023). For the other datasets, we used the entire test set for evaluation. We used the math-evaluation-harness library (Gou and Zhang, 2024) to ensure consistency and comparability with existing work, following the evaluation framework of DeepSeek-Math (Zhihong Shao, 2024). We used the in-context examples provided by the math-evaluation-harness for few-shot chain-of-thought (CoT) prompting and report accuracy as the performance metric.

**Baselines** We benchmarked our method against both single LM decoding algorithms and LM ensemble approaches. We evaluated two reward-free

| Category | Base LLM | Method | GSM8K | MATH | SVAMP | ASDiv | MQA | Average |
|---|---|---|---|---|---|---|---|---|
| Single LLM | LLaMA-3 | Greedy | 69.4 | 12.0 | 81.2 | 77.9 | 21.4 | 52.4 |
| | | SC | 69.3 | 11.8 | 79.5 | 76.4 | 18.9 | 51.2 |
| | | BS | 74.2 | 19.0 | 81.0 | 79.8 | 21.7 | 55.1 |
| | | BoN | 74.6 | 13.4 | 83.3 | 77.7 | 16.6 | 53.1 |
| | Gemma-2 | Greedy | 80.9 | 40.4 | 69.2 | 65.6 | 27.9 | 56.8 |
| | | SC | 80.6 | 39.4 | 68.1 | 66.2 | 27.0 | 56.3 |
| | | BS | 81.4 | 40.8 | 67.3 | 67.2 | 28.6 | 57.1 |
| | | BoN | 82.7 | 41.6 | 73.2 | 69.5 | 29.1 | 59.2 |
| | DeepSeek-Math | Greedy | 46.6 | 28.6 | 64.0 | 70.6 | 63.8 | 54.7 |
| | | SC | 47.1 | 27.8 | 60.2 | 68.0 | 60.9 | 52.8 |
| | | BS | 52.4 | 29.0 | 60.1 | 67.5 | 66.8 | 55.2 |
| | | BoN | 65.9 | 35.0 | 73.0 | 83.5 | 66.1 | 64.7 |
| | Rho-Math | Greedy | 67.6 | 29.6 | 76.6 | 77.8 | 55.8 | 61.5 |
| | | SC | 66.9 | 28.2 | 74.2 | 77.3 | 57.5 | 60.8 |
| | | BS | 69.9 | 28.8 | 77.7 | 81.1 | 58.2 | 63.1 |
| | | BoN | 74.8 | 34.6 | 79.8 | 82.2 | 61.6 | 66.6 |
| Ensemble | Top-3 | BoE | 80.0 | 36.0 | **84.5** | 83.8 | 65.1 | 69.9 |
| | Top-3 | EBS | 66.7 | 41.0 | 80.8 | 78.2 | 64.0 | 66.1 |
| | All | Blender † | 51.9 | 1.4 | 71.3 | 69.0 | 21.9 | 43.1 |
| | Top-3 | MoA † | 42.5 | 22.2 | 44.3 | 47.4 | 60.4 | 43.4 |
| | All | EVA † | 66.3 | 26.0 | 73.8 | 81.4 | 54.6 | 60.4 |
| | Top-3 | Ours | **84.1 (+1.4)** | **45.2 (+3.6)** | 84.0 (-0.5) | **84.4 (+0.6)** | **71.1 (+4.3)** | **73.8 (+3.9)** |

Table 2: **Summary of main results.** We measure the accuracy on the test set of five math reasoning benchmarks. We also report the average of the performances on five datasets in the rightmost column. We highlight the best model in **bold** and the second-best model with an underline, respectively. †: we reuse the official code for experiments.

algorithms, greedy decoding and self-consistency (SC) (Wang et al., 2023), as well as two process reward-guided algorithms, Best-of-N (BoN) (Lightman et al., 2023) and Beam Search (BS) (Yu et al., 2024), as baselines for single LMs. We also compared our method with the token-level ensemble approach EVA (Xu et al., 2024), as well as the output-level ensemble methods LLM-Blender (Jiang et al., 2023b) and MoA (Wang et al., 2024a).

Furthermore, we propose two novel variations of process reward-guided decoding algorithms by leveraging multiple LLMs. Specifically, Best-of-Ensemble (BoE) selects the highest-rewarded one among the complete solutions generated by various LLMs as the final output. Ensemble Beam Search (EBS) generates candidate reasoning steps using multiple LLMs instead of a single LLM and performs beam search. Further details of the baselines are provided in Appendix D.

**Implementation Details** We considered two general domain LLMs, LLaMA-3 8B (LLaMA, 2024) and Gemma-2 9B (Gemma, 2024), and two math LLMs, DeepSeek-Math 7B (Zhihong Shao, 2024) and Rho-Math 7B (Lin et al., 2024), as base models for the model ensemble. We set $n_{\text{iter}} = 200$ and perform LM random selection without replacement for all experiments, except when the specific hyperparameter setting is mentioned. After running LE-MCTS on each example, we extracted all trajectories in the MCTS tree that reached the terminal node. We then ranked these trajectories using PRM and selected the top-ranked trajectory as the final output. We tested two sets of base models for ensemble methods, Top-3 and All, and report the one with the better average performance as the main result. Specifically, for Top-3, a distinct set of LLMs was used for each dataset, while for All, we ensembled all four base LLMs. Full experimental results and further details of the base model selection for Top-3 are provided in Appendix A.

### 3.2 Main Results

We report the main results on math benchmarks in Table 2.

**LE-MCTS outperforms or matches existing approaches.** The results show that LE-MCTS matches the performance of the best model on SVAMP and outperforms all other models in the remaining datasets. Specifically, the average performance across the five tasks is 3.9% higher than the second-best model, BoE. This suggests that LE-MCTS is a versatile language model ensemble framework for complex reasoning, particularly in mathematical problem-solving.

**LE-MCTS is especially good at complex reasoning.** LE-MCTS significantly outperforms the

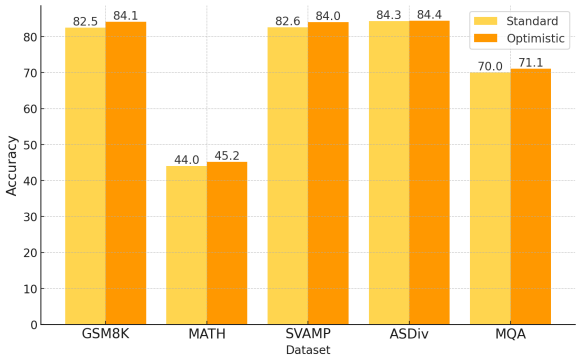


Figure 3: Ablation study on value backpropagation strategies.

second-best model on MATH and MQA. MATH consists of competition-level math problems, while MQA includes GRE and GMAT exam questions, both of which require more diverse and complex reasoning skills than other grade-school math datasets. These challenging problems often involve longer reasoning steps because of the need for in-depth reasoning.

What distinguishes LE-MCTS from other approaches is its ability to prioritize in-depth reasoning. For example, the full-expansion criterion guides LE-MCTS to explore deeper reasoning steps rather than expanding the breadth of early-stage reasoning. Moreover, the small UCT constant $C$ encourages the selection of high-value nodes over underexplored nodes. When combined with optimistic backpropagation, this setting also makes LE-MCTS favor deeper exploration. In contrast, BoE and EBS rely on nucleus sampling to generate reasoning steps, which does not guarantee deeper or more thorough reasoning.

**Process-level ensembles outperform token- and output-level approaches in math reasoning.** Token-level and output-level ensemble approaches perform significantly worse than process-level ensemble methods. In some cases, they even perform worse than the weakest single LLM baseline. For instance, LLM-Blender performs poorly on MATH, achieving an accuracy of only 1.4%. We believe that GenFuser, which aggregates outputs from multiple LLMs to generate the final result, has limited capacity for comprehending complex reasoning. In contrast, MoA employs a larger language model for output fusion and utilizes prompt engineering to enhance aggregation. Consequently, its accuracy on MATH surpasses that of LLM-Blender, though it still falls short of process-level ensemble methods.

| $C$ | GSM8K | MATH | SVAMP | ASDiv | MQA |
|---|---|---|---|---|---|
| 0.5 | 81.7 | **45.2** | 82.7 | 84.2 | **71.1** |
| 1.0 | 83.7 | 43.6 | **84.0** | **84.4** | 69.0 |
| 1.414 | **84.1** | 44.4 | 83.8 | 84.2 | 68.6 |

Table 3: Effect of the UCT constant $C$.

EVA demonstrates better average performance than output-level ensemble baselines. However, its vocabulary alignment requires matching embedding dimensions across the base models, which limits its applicability. As a result, we cannot use Gemma-2 as the base model for EVA, even though it achieves the highest performance among the base LLMs on GSM8K and MATH. In contrast, process-based algorithms, including ours, are not constrained by architecture or vocabulary, ensuring broad applicability across various LLMs.

### 3.3 Analysis

**Backpropagation Strategy** As an ablation study, we compared LE-MCTS with standard value backpropagation to LE-MCTS with optimistic value backpropagation. The results in Fig. 3 show that the optimistic backpropagation consistently improves performance across all datasets, with increases ranging from 0.1% to 1.6%. Moreover, as shown in Appendix B, we observed higher average rewards for leaf nodes when using optimistic backpropagation. This occurs because optimistic backpropagation updates only the values of the highest-value nodes through the tree, thereby neglecting low-value nodes during the search. As PRM more accurately estimates the process rewards, the performance gap between optimistic and standard strategies is expected to widen.

**UCT Constant, $C$** The constant $C$ controls the balance between exploration and exploitation in LE-MCTS. In our framework, the depth of the node corresponds to the number of reasoning steps. The experimental results in Appendix C show that the average depth of the leaf nodes tends to increase as $C$ decreases. We hypothesize that simpler math problems, such as grade-school math word problems (*e.g.*, SVAMP and ASDiv), do not require long reasoning chains and benefit from a higher $C$, as it facilitates more effective reasoning in the early steps, such as identifying mathematical variables. Conversely, more challenging math problems, such as those in MATH and MQA, require competition-level reasoning skills. In these cases, a lower $C$

| $n_{iter}$ | GSM8K | MATH | SVAMP | ASDiv | MQA |
|---|---|---|---|---|---|
| 10 | 79.8 | 43.8 | 82.9 | 82.4 | 65.7 |
| 25 | 81.0 | 43.4 | 82.9 | 83.5 | 65.5 |
| 50 | 81.5 | 44.4 | 82.7 | 83.2 | 68.9 |
| 100 | 82.9 | 45.2 | 83.1 | 83.5 | 68.2 |
| 200 | **84.1** | **45.2** | **84.0** | **84.4** | **71.1** |

Table 4: Effect of the maximum number of MCTS iterations $n_{iter}$.

| Method | ASDiv | | MATH | |
|---|---|---|---|---|
| | VRAM (↓) | min/ex (↓) | VRAM (↓) | min/ex (↓) |
| BoE | 76.7 | 17.6 | 64.8 | 71.1 |
| EBS | 79.2 | 12.3 | 71.8 | 47.2 |
| Blender | 70.4 | 22.2 | 66.5 | 59.1 |
| MoA | 67.4 | 84.4 | 79.8 | 93.7 |
| EVA | 69.9 | 92.2 | 70.3 | 480.2 |
| Ours$_{n_{iter}=25}$ | 76.7 | 34.6 | 64.6 | 129.1 |
| Ours$_{n_{iter}=200}$ | 77.4 | 112.2 | 71.0 | 342.2 |

Table 5: **Efficiency analysis.** We compare the efficiency of our method with existing ensemble approaches based on peak VRAM usage and throughput. Throughput is measured as the average time per example, reported in minutes per example (min/ex). VRAM usage is quantified as the maximum value observed during inference, expressed in gigabytes (GB). For both metrics, lower values indicate higher efficiency.

is advantageous, as it encourages a deeper exploration of reasoning paths instead of focusing on the early steps.

To test our hypothesis, we evaluated LE-MCTS with $C \in \{0.5, 1.0, 1.414\}$[1], and the results are reported in Table 3. The results support our hypothesis that $C = 0.5$ yields the best performance on MATH and MQA, while $C = 1.0$ and $C = 1.414$ outperform $C = 0.5$ on GSM8K, SVAMP, and ASDiv. Our experiment offers a guide for choosing an appropriate $C$ value for previously unseen problems. For problems with a complexity level comparable to simple grade-school math word problems, a $C$ value greater than 1 is recommended to facilitate effective exploration of early-stage reasoning steps. In contrast, for more complex problems, a $C$ value below 1 is preferable, as it encourages a more thorough investigation of the in-depth reasoning paths.

**Maximum Number of MCTS Iterations,** $n_{iter}$ The hyperparameter $n_{iter}$ plays a crucial role in determining the balance between performance and computational cost. A large $n_{iter}$ increases the execution time of the algorithm, while a too small $n_{iter}$ may be insufficient to find an optimal reasoning path. To investigate the effect of $n_{iter}$ on performance, we varied $n_{iter}$ over the set $\{10, 25, 50, 100, 200\}$ and report the results in Table 4. The results show that performance improves as $n_{iter}$ increases from 10 to 200, suggesting that we can find a better reasoning path as we run more MCTS iterations. However, this improvement varies across datasets, and performance generally saturates beyond 200 iterations. These results highlight the importance of selecting an appropriate $n_{iter}$ to balance performance with computational cost.

**Efficiency Analysis** We follow Dehghani et al. (2022) to compare the efficiency of the proposed method with existing ensemble approaches. Specifically, we analyze efficiency along two axes: GPU memory consumption and throughput. Throughput is defined as the average time required to process each example. We use peak VRAM usage during inference as the memory consumption metric. We choose VRAM usage as an efficiency metric because the number of floating-point operations (FLOPs), although commonly used, is difficult to compare fairly across different decoding algorithms. In contrast, VRAM usage serves as an indirect indicator of the number of parameters and the input sequence length involved in the decoding process, which are the most important factors for measuring computational cost in Transformer architectures.

For a fair comparison, we measured VRAM usage and throughput on a single NVIDIA H100 80GB GPU, using the same CUDA and PyTorch versions. We evaluated these efficiency metrics on the most complex dataset, MATH, and the simplest dataset, ASDiv. The results in Table 5 show that EBS is the most efficient algorithm in terms of throughput. For peak VRAM usage, MoA and LE-MCTS$_{n_{iter}=25}$ are the most efficient algorithms on ASDiv and MATH, respectively. Both output- and token-level ensemble methods show high throughput in MATH, while LLM-Blender and MoA suffer from the long input sequence length of the aggregator, and EVA lacks optimization for generating long sequence in its implementation.

Our LE-MCTS with $n_{iter} = 200$ is more resource-intensive than most existing algorithms but achieves better performance. While LE-MCTS offers significant performance improvements on

[1]We follow the standard practice (Kocsis and Szepesvári, 2006; Browne et al., 2012) to choose the value $\sqrt{2}$, along with smaller values of 0.5 and 1.0 for greater exploitation.

complex problems, its advantages on simpler tasks are marginal. In ASDiv, LE-MCTS is less efficient than BoE and EBS in throughput, even when we run a small number of MCTS iterations $n = 25$. This results underscore that while LE-MCTS's in-depth search is beneficial for solving complex reasoning problems, it may introduce unnecessary computational overhead for straightforward tasks, where EBS and BoE can achieve comparable performance at a lower cost. Therefore, LE-MCTS is particularly well-suited for challenging reasoning tasks that require in-depth analysis, where a trade-off between performance and computational cost is acceptable.

# 4 Related Work

## 4.1 Language Model Ensemble

Ensemble learning is a widely used approach to improve model performance by integrating multiple weaker models (Lu et al., 2024b). In large language models (LLMs), ensemble methods have been typically applied at either the token level or the output level. Token-level ensembling (Liu et al., 2021, 2024a; Li et al., 2024; Mavromatis et al., 2024) has involved merging token logits or probabilities from multiple LLMs, which often requires that the models share the same token vocabulary and model architecture. To overcome this limitation, EVA (Xu et al., 2024) have mapped the output distributions of different LLMs into a unified space through pre-trained mapping. In contrast, output-level ensemble methods (Jiang et al., 2023b; Wang et al., 2024a; Izacard and Grave, 2021; Ravaut et al., 2022) have combined entire model outputs using an additional fusion model. For example, LLM-Blender (Jiang et al., 2023b) have introduced a general framework that employs a pair ranker to filter the top K optimal outputs before merging them through a fusion model to generate the final result.

Our work introduces a novel process-level ensemble method that offers greater flexibility than token-level approaches, as LE-MCTS does not need to match the vocabulary or architecture. Moreover, in output-level ensemble methods, the aggregator receives the outputs of multiple LLMs as input, which can cause the input sequence length to exceed the limit it can handle. In contrast, the maximum input sequence length of LE-MCTS remains the same as when solving a math reasoning problem with a single LLM using greedy decoding. Therefore, output-level ensemble methods are not suitable for complex reasoning tasks that require long, in-depth reasoning.

## 4.2 Reward-Guided Decoding

Two main types of verifiers for mathematical reasoning problems are the Outcome Reward Model (ORM) and the Process Reward Model (PRM). ORM assesses the entire solution, whereas PRM evaluates individual steps, providing more granular feedback. Both approaches have shown improvements in mathematical reasoning compared to self-consistency (Wang et al., 2023), yet evidence has shown that PRM outperforms ORM (Lightman et al., 2023; Wang et al., 2024b). Previous studies have used rewards to guide the decoding process at the sentence level (Uesato et al., 2022; Welleck et al., 2022; Yao et al., 2024) and the token level (Dathathri et al., 2020; Yang and Klein, 2021; Chaffin et al., 2022; Li et al., 2023). ARGS (Khanov et al., 2024) has incorporated language model alignment as a reward-guided search. Yu et al. (2024) and Ma et al. (2023) have applied process rewards to enhance heuristic search algorithms. Recently, ReST-MCTS* (Zhang et al., 2024) has also employed process reward-guided MCTS, similar to our approach, to generate high-quality data for self-training. However, our use of MCTS focuses on ensembling LLMs. To the best of our knowledge, this is the first attempt to apply process-reward guidance for LLM ensembling.

# 5 Conclusion

In this paper, we introduced a novel framework for process-level ensembling of language models, addressing key limitations of traditional language model ensemble methods. We formulated the step-by-step reasoning with an ensemble of language models as a Markov decision process. By leveraging an existing process-based reward model and Monte Carlo Tree Search, our approach effectively navigates the unified space of reasoning steps generated by different language models, discovering more accurate solutions. Extensive experiments on mathematical reasoning benchmarks demonstrate the efficacy of the proposed method, especially in solving complex reasoning problems. We believe this approach paves the way for broader process-level ensemble of language models, as it can be applied to any step-by-step reasoning problem where an appropriate process-reward model exists.

## Limitations

Our proposed LE-MCTS framework relies on signals from the process-based reward model to effectively navigate the space of reasoning steps. When the PRM fails to compute rewards accurately, LE-MCTS also fails. While Math-Shepherd's PRM generally performs well on established mathematical datasets, there is no guarantee that it will be applicable to novel math problems. Moreover, PRMs for other complex, step-by-step reasoning tasks remain relatively unexplored. Therefore, developing robust and generalizable PRMs is crucial for enhancing LE-MCTS, and we believe this is a valuable future direction.

Another limitation of our work is the need to select base models. The experimental results in Appendix A suggest that using weak base models harms the performance of LE-MCTS. Although we demonstrated that even a small amount of synthetic data can effectively identify weak base models, this approach may not generalize well to other tasks and datasets. Nevertheless, selecting base models provides a clear advantage in terms of efficiency. By ensembling only a small number of LLMs, LE-MCTS achieves greater efficiency in both VRAM usage and the number of MCTS iterations. We believe a robust selection algorithm for identifying effective base models is another important future direction.

## References


Aida Amini, Saadia Gabriel, Shanchuan Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. 2019. MathQA: Towards interpretable math word problem solving with operation-based formalisms. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2357–2367, Minneapolis, Minnesota. Association for Computational Linguistics.

Richard Bellman. 1957. A markovian decision process. *Journal of Mathematics and Mechanics*, 6(5):679–684.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, et al. 2024. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 17682–17690.

Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. 2012. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43.

Antoine Chaffin, Vincent Claveau, and Ewa Kijak. 2022. PPL-MCTS: Constrained textual generation through discriminator-guided MCTS decoding. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2953–2967, Seattle, United States. Association for Computational Linguistics.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. 2022. Scaling instruction-finetuned language models. *arXiv preprint*.

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

Rémi Coulom. 2006. Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, pages 72–83. Springer.

Prakhar Dathathri, Andrea Madotto, Janice Lan, Jane Hung Liu, Asli Celikyilmaz, Amir Zadeh, and Hao Cheng Poon. 2020. Plug and play language models: A simple approach to controlled text generation. In *International Conference on Learning Representations (ICLR)*.

Mostafa Dehghani, Yi Tay, Anurag Arnab, Lucas Beyer, and Ashish Vaswani. 2022. The efficiency misnomer. In *International Conference on Learning Representations (ICLR)*.

António Farinhas, José de Souza, and Andre Martins. 2023. An empirical study of translation hypothesis ensembling with large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 11956–11970, Singapore. Association for Computational Linguistics.

Xidong Feng, Ziyu Wan, Muning Wen, Ying Wen, Weinan Zhang, and Jun Wang. 2023. Alphazero-like tree-search can guide large language model decoding and training. *arXiv preprint arXiv:2309.17179*.

Gemini. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *Preprint*, arXiv:2403.05530.

Gemma. 2024. Gemma 2: Improving open language models at a practical size. *Preprint*, arXiv:2408.00118.

Zhibin Gou and Yue Zhang. 2024. Llm math evaluation harness: A toolkit for benchmarking llms on mathematical reasoning tasks. https://github.com/ZubinGou/math-evaluation-harness. MIT License.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2021. Deberta: Decoding-enhanced bert with disentangled attention. In *International Conference on Learning Representations*.

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

Gautier Izacard and Edouard Grave. 2021. Leveraging passage retrieval with generative models for open domain question answering. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 874–880, Online. Association for Computational Linguistics.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023a. Mistral 7b. *Preprint*, arXiv:2310.06825.

Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. 2023b. LLM-blender: Ensembling large language models with pairwise ranking and generative fusion. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14165–14178, Toronto, Canada. Association for Computational Linguistics.

Maxim Khanov, Jirayu Burapacheep, and Yixuan Li. 2024. Args: Alignment as reward-guided search. In *International Conference on Learning Representations (ICLR)*.

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

Tianlin Li, Qian Liu, Tianyu Pang, Chao Du, Qing Guo, Yang Liu, and Min Lin. 2024. Purifying large language models by ensembling a small language model. *Preprint*, arXiv:2402.14845.

Xiang Lisa Li, Ari Holtzman, Daniel Fried, Percy Liang, Jason Eisner, Tatsunori Hashimoto, Luke Zettlemoyer, and Mike Lewis. 2023. Contrastive decoding: Open-ended text generation as optimization. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12286–12312, Toronto, Canada. Association for Computational Linguistics.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. *arXiv preprint arXiv:2305.20050*.

Zhenghao Lin, Zhibin Gou, Yeyun Gong, Xiao Liu, Yelong Shen, Ruochen Xu, Chen Lin, Yujiu Yang, Jian Jiao, Nan Duan, and Weizhu Chen. 2024. Rho-1: Not all tokens are what you need. *Preprint*, arXiv:2404.07965.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. 2017. Program induction by rationale generation: Learning to solve and explain algebraic word problems. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 158–167, Vancouver, Canada. Association for Computational Linguistics.

Alisa Liu, Xiaochuang Han, Yizhong Wang, Yulia Tsvetkov, Yejin Choi, and Noah A. Smith. 2024a. Tuning language models by proxy. *Preprint*, arXiv:2401.08565.

Alisa Liu, Maarten Sap, Ximing Lu, Swabha Swayamdipta, Chandra Bhagavatula, Noah A. Smith, and Yejin Choi. 2021. DExperts: Decoding-time controlled text generation with experts and anti-experts. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 6691–6706, Online. Association for Computational Linguistics.

Cong Liu, Xiaojun Quan, Yan Pan, Liang Lin, Weigang Wu, and Xu Chen. 2024b. Cool-fusion: Fuse large language models without training. *Preprint*, arXiv:2407.19807.

LLaMA. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Jianqiao Lu, Zhiyang Dou, Hongru Wang, Zeyu Cao, Jianbo Dai, Yingjia Wan, Yinya Huang, and Zhijiang Guo. 2024a. Autocv: Empowering reasoning with automated process labeling via confidence variation. *arXiv preprint arXiv:2405.16802*.

Jinliang Lu, Ziliang Pang, Min Xiao, Yaochen Zhu, Rui Xia, and Jiajun Zhang. 2024b. Merge, ensemble, and cooperate! a survey on collaborative strategies in the era of large language models. *Preprint*, arXiv:2407.06089.

Qianli Ma, Haotian Zhou, Tingkai Liu, Jianbo Yuan, Pengfei Liu, Yang You, and Hongxia Yang. 2023. Let's reward step by step: Step-level reward model as the navigators for reasoning. *arXiv preprint arXiv:2310.10080*.

Costas Mavromatis, Petros Karypis, and George Karypis. 2024. Pack of llms: Model fusion at test-time via perplexity optimization. *arXiv preprint arXiv:2404.11531*.

Shen-yun Miao, Chao-Chun Liang, and Keh-Yih Su. 2020. A diverse corpus for evaluating and developing English math word problem solvers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 975–984, Online. Association for Computational Linguistics.

OpenAI. 2024. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are NLP models really able to solve simple math word problems? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2080–2094, Online. Association for Computational Linguistics.

Mathieu Ravaut, Shafiq Joty, and Nancy Chen. 2022. Towards summary candidates fusion. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 8488–8504, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. 2017. Mastering chess and shogi by self-play with a general reinforcement learning algorithm.

Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. 2020. Learning to summarize with human feedback. In *Advances in Neural Information Processing Systems*, volume 33, pages 3008–3021. Curran Associates, Inc.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. 2022. Solving math word problems with process-and outcome-based feedback. *arXiv preprint arXiv:2211.14275*.

Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Zou. 2024a. Mixture-of-agents enhances large language model capabilities. *arXiv preprint arXiv:2406.04692*.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. 2024b. Math-shepherd: Verify and reinforce LLMs step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9426–9439, Bangkok, Thailand. Association for Computational Linguistics.

Zhong-Ling Wang, Po-Hsien Huang, Wen-Yau Hsu, and Hen-Hsen Huang. 2023. Self-adapted utterance selection for suicidal ideation detection in lifeline conversations. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 1436–1446, Dubrovnik, Croatia. Association for Computational Linguistics.

Sean Welleck, Jiacheng Liu, Ximing Lu, Hannaneh Hajishirzi, and Yejin Choi. 2022. Naturalprover: Grounded mathematical proof generation with language models. *Advances in Neural Information Processing Systems*, 35:4913–4927.

Zichen Wu, Hsiu-Yuan Huang, Fanyi Qu, and Yunfang Wu. 2024. Mixture-of-prompt-experts for multimodal semantic understanding. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 11381–11393, Torino, Italia. ELRA and ICCL.

Yangyifan Xu, Jinliang Lu, and Jiajun Zhang. 2024. Bridging the gap between different vocabularies for LLM ensemble. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 7140–7152, Mexico City, Mexico. Association for Computational Linguistics.

Kevin Yang and Dan Klein. 2021. FUDGE: Controlled text generation with future discriminators. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3511–3535, Online. Association for Computational Linguistics.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2024. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36.

Fei Yu, Anningzhe Gao, and Benyou Wang. 2024. OVM, outcome-supervised value models for planning in mathematical reasoning. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 858–875, Mexico City, Mexico. Association for Computational Linguistics.

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. 2023. Scaling relationship on learning mathematical reasoning with large language models. *arXiv preprint arXiv:2308.01825*.

Dan Zhang, Sining Zhoubian, Yisong Yue, Yuxiao Dong, and Jie Tang. 2024. Rest-mcts*: Llm self-training via process reward guided tree search. *arXiv preprint arXiv:2406.03816*.

Qihao Zhu Runxin Xu Junxiao Song Mingchuan Zhang Y.K. Li Y. Wu Daya Guo Zhihong Shao, Peiyi Wang. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models.

| Model | GSM8K | MATH | SVAMP | ASDiv | MQA |
|---|---|---|---|---|---|
| Rho-Math | 81.2 | 56.2 | 87.5 | 100 | 62.5 |
| LLaMA-3 | 81.2 | 31.2 | 87.5 | 100 | 25 |
| Gemma-2 | 87.5 | 81.2 | 37.5 | 100 | 37.5 |
| DeepSeek-Math | 56.2 | 43.8 | 87.5 | 87.5 | 87.5 |

Table 6: Performance on 16 synthetic examples.

# A Base Model Selection

LLMs sometimes exhibit significantly lower performance on certain datasets, such as LLaMA-3 on MATH and MQA. We hypothesize that including such LLMs as base models may harm the performance of ensemble frameworks. However, selecting base models for unseen problems is challenging because we cannot measure their performance in advance. To address this, we generated synthetic examples that mirror the original math datasets in both difficulty and style, and we used the performance on these synthetic data for base model selection. Specifically, we instructed GPT-4o to generate 16 synthetic examples for each dataset, as shown in Figs. 5-9. We evaluated the greedy decoding performance of each LLM on the synthetic data and selected the top three models as base models for the ensemble. The evaluation results on the synthetic data are presented in Table 6.

For the selection of top-3 models, we heuristically included high-performing LLMs and excluded low-performing ones. For example, although DeepSeek-Math performs worse than the other models on ASDiv, it still demonstrates strong performance. Therefore, we included all models as base models. In contrast, the performance gap between the second and third best models on MQA was substantial, so we selected only Rho-Math and DeepSeek-Math as base models. For GSM8K, MATH, and SVAMP, we used the top-3 models as base models. Note that EVA was tested only with All, as it cannot utilize Gemma-2 due to a mismatch in hidden dimensions with the other LLMs.

We present the complete results for the Top-3 and All sets of base models in Table 7. Except for LLM-Blender, Top-3 consistently shows better average performance across all ensemble frameworks. Moreover, the performance gap between Top-3 and All in LLM-Blender is minimal or zero. These results indicate that weak base models indeed harm the performance of ensemble frameworks and that synthetic examples can effectively identify weak base models.

| Method | Base Model | GSM8K | MATH | SVAMP | ASDiv | MQA |
|---|---|---|---|---|---|---|
| EBS | Top-3 | 66.7 | 41.0 | 80.8 | 78.2 | 64.0 |
| | All | 54.6 | 41.8 | 78.8 | 78.2 | 61.3 |
| BoE | Top-3 | 80.0 | 36.0 | 84.5 | 83.8 | 65.1 |
| | All | 79.4 | 29.0 | 85.1 | 83.8 | 51.2 |
| MoA | Top-3 | 42.5 | 22.2 | 44.3 | 47.4 | 60.4 |
| | All | 43.3 | 10.4 | 48.4 | 47.4 | 61.2 |
| Blender | Top-3 | 49.9 | 1.4 | 69.0 | 69.0 | 21.9 |
| | All | 51.9 | 1.4 | 71.3 | 69.0 | 21.9 |
| LE-MCTS | Top-3 | 84.1 | 45.2 | 84.0 | 84.4 | 71.1 |
| | All | 84.2 | 40.0 | 79.7 | 84.4 | 67.5 |

Table 7: Full experimental results of language model ensemble approaches with two base model configurations: Top-3 and All.

# B Average Reward Distribution of Leaf Nodes

We measured the average reward received by leaf nodes in MCTS for optimistic and standard backpropagation strategies. Specifically, we first measured the process rewards of all leaf nodes in the tree, then computed the average per sample and reported the results in the KDE plots Figs. 10-14. The value in parentheses represents the mean of all points in the plot.

For complex math problems such as MATH and MQA, leaf nodes received relatively lower rewards compared to simpler problems. For easy math problems, like ASDiv, the distribution is left-skewed, with the probability mass concentrated in the range [0.8, 1.0]. Leaf nodes consistently received higher rewards when using optimistic value backpropagation, regardless of the dataset. These results indicate that optimistic backpropagation is more effective than standard backpropagation in discovering high-reward reasoning trajectories. Furthermore, LE-MCTS's preference for high-reward reasoning paths guarantees performance improvements, particularly when paired with a better PRM.

# C Average Depth Distribution of Leaf Nodes

Similar to Appendix B, we measured the average depth of leaf nodes for $C = \{0.5, 1.0, 1.414\}$. Specifically, we first measured the depth of all leaf nodes in the tree, then computed the average per sample and reported the results in the KDE plots Figs. 15-19. The value in parentheses represents the mean of all points in the plot. The results show that the length of reasoning trajectories increases as $C$ decreases. These results support our claim that LE-MCTS tends to perform more in-depth reasoning when a lower UCT constant $C$ is used.

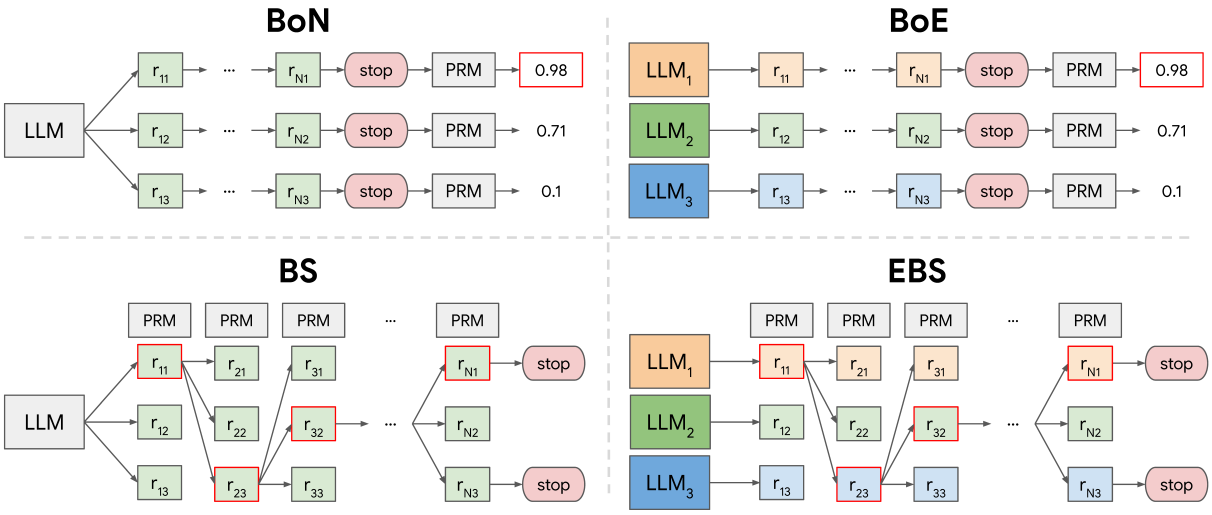


Figure 4: An illustration of process reward-guided decoding algorithms with $N = 3$ and a beam size of 1.

## D Detailed Explanation of Baselines

In this section, we briefly describe the baseline methods to make our paper self-contained.

**Process reward-guided decoding algorithms for a single LLM and ensemble language models** Best-of-N (BoN) (Stiennon et al., 2020; Lightman et al., 2023) is a popular algorithm for aligning language models to human preferences. During inference, $N$ samples are drawn from the language model, and the sample with the highest reward, determined by a reward model, is returned as the final output. BoN has also been applied to math problems (Lightman et al., 2023; Yuan et al., 2023), demonstrating its effectiveness. Process reward-guided Beam Search (BS) was first proposed in OVM (Ma et al., 2023), which has employed a beam search strategy guided by OVM. Unlike conventional beam search, which relies on token-level probability, BS is steered by the estimated process reward at each step. We followed the BS implementation in OVM but used PRM from Math-Shepherd as the reward model, instead of OVM, to compute a consistent reward signal across all baselines.

Best-of-Ensemble (BoE) and Ensemble Beam Search (EBS) are simple modifications of BoN and BS, as depicted in Fig. 4. For BoE, instead of using a single LLM, we employ multiple LLMs to generate candidates and select the best one among them. For EBS, at each step, multiple LLMs generate candidates for beam search. These straightforward modifications of existing process-reward guided decoding algorithms are significantly more effective than the original methods, as they provide a wider search space by incorporating different LLMs. For all models, we set $N = 9$ and sample candidates using nucleus sampling with a temperature 0.5.

**LLM-Blender** (Jiang et al., 2023b) LLM-Blender consists of two modules: PairRanker and GenFuser. PairRanker employs a specialized pairwise comparison method to distinguish subtle differences between candidate outputs. It jointly encodes the input text and a pair of candidates, using cross-attention encoders to determine the superior one. Then, GenFuser aims to merge the top-ranked candidates, generating an improved output by capitalizing on their strengths and mitigating their weaknesses. PairRanker employs DeBERTa (He et al., 2021) and GenFuser employs Flan-T5-XL (Chung et al., 2022) as the backbone and both are fine-tuned on the MixInstruct dataset proposed in LLM-Blender.

**Mixture-of-Agents (MoA)** (Wu et al., 2024) MoA employs multiple LLMs to iteratively enhance generation quality. It constructs a layered architecture in which each layer consists of multiple LLM agents. Specifically, LLMs in the first layer, denoted as agents $A_{1,1}, \ldots, A_{1,n}$, independently generate responses to a given prompt. These responses are then passed on to the next layer of agents $A_{2,1}, \ldots, A_{2,n}$, which can reuse models from the first layer, for further refinement. This iterative refinement process is repeated over several cycles to produce a more robust and comprehensive response. In the final layer, a primary aggregator synthesizes all the outputs to obtain a final solution.

**Ensemble LLMs via Vocabulary Alignment (EVA)** (Xu et al., 2024) EVA proposed a novel vocabulary alignment method and fused LLMs at the token level. Their approach is based on the observation that, although various LLMs have distinct vocabularies, they often share a significant number of overlapping tokens. EVA leverages these tokens as bridges by first extracting embeddings of the overlapping tokens and learning a mapping matrix to project these embeddings into a shared space. Subsequently, by computing similarity scores between tokens in these vocabularies, EVA derives the semantic projection matrix $W$. This enables the projection of output distributions from one LLM to another, allowing for the generation of reasonable tokens based on the fused distribution of these LLMs at each inference step. Finally, they further enhance the approach by devising a filtering strategy to exclude models that generate unfaithful tokens.

## E Tasks and Dataset Statistics

**GSM8K** (Cobbe et al., 2021) Grade School Math 8K (GSM8K) is a dataset of 8.5K high-quality linguistically diverse grade school math word problems. These problems take between 2 and 8 steps to solve. Solutions primarily involve performing a sequence of elementary calculations using basic arithmetic operations to reach the final answer. We used 1,319 test examples for the experiment.

**MATH** (Hendrycks et al., 2021) MATH is a dataset of 12.5k challenging competition mathematics problems. Each problem in MATH has a full step-by-step solution and requires a wide range of mathematical problem solving abilities at varying levels of difficulty and topics. Although the original MATH dataset provides 5,000 test examples, PRM800K (Lightman et al., 2023) used 4,500 of these for training. Following standard practice (Lightman et al., 2023; Wang et al., 2024b), we used the remaining 500 examples (MATH500) for evaluation to prevent data leakage.

**SVAMP** (Patel et al., 2021) Simple Variations on Arithmetic Math word Problems (SVAMP) is a dataset of 1,000 arithmetic word problems with grade level up to 4 by applying simple variations over word problems in an existing dataset. The authors modified the ASDiv-A (Miao et al., 2020) dataset with three variations: changing the question object and structure, tweaking the underlying reasoning, and shuffling objects and phrases. We used the entire dataset for the evaluation.

**ASDiv** (Miao et al., 2020) Academia Sinica Diverse MWP Dataset (ASDiv) is a dataset of 2,305 math word problems commonly taught in elementary school. ASDiv focuses on the diversity of math problems in terms of text patterns and problem types. We used the entire dataset for the evaluation.

**MQA** (Amini et al., 2019) MathQA (MQA) is a dataset of 37k English multiple choice math word problems that cover multiple math domain categories by modeling operation programs corresponding to word problems in the AQuA dataset (Ling et al., 2017). Specifically, MQA manually annotated GRE- and GMAT-level math word problems in AQuA with formal operation programs. We used the 1,000 examples provided by the math-evaluation-harness library for the evaluation.

## F Case Study

We present an example generated by LE-MCTS and BoE using a test case from GSM8K in Table 8. BoE, the second-best ensemble model on GSM8K, made an error in the intermediate reasoning step, "*So the new ... $50,000.*". A similar mistake appears in the trajectory discovered by LE-MCTS, despite receiving a high reward of 0.755. However, the highest-reward reasoning chain in LE-MCTS (*i.e.*, the path that reaches the leaf node with a reward of 0.932) demonstrates perfect reasoning. The provided example illustrates the flexibility of LE-MCTS in reasoning. Although LE-MCTS can make mistakes during the search, it is capable of identifying better reasoning paths by incorporating a lookahead search and diverse reasoning steps from multiple LLMs.

| |
|---|
| **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? |
| **LE-MCTS, Correct answer** (*reward*: 0.932)<br><br>*LLaMA-3*: Let's break it down step by step!<br>*Gemma-2*: 1. **Calculate the total investment:** Josh spent $80,000 (house) + $50,000 (repairs) = $130,000<br>*Gemma-2*: 2. **Calculate the increase in value:** A 150% increase means the house's value went up by 1.5 times its original price. So, the increase is $80,000 * 1.5 = $120,000<br>*Gemma-2*: 3. **Calculate the new value:** The house is now worth $80,000 (original) + $120,000 (increase) = $200,000<br>*Rho-Math*: 4. **Calculate the profit:** The profit is the difference between the new value and the total investment. So, $200,000 - $130,000 = $70,000<br>*LLaMA-3*: 5. **Answer:** Josh made a profit of $70,000. |
| **LE-MCTS, Wrong answer** (*reward*: 0.755)<br><br>*Gemma-2*: Here's how to solve the house flipping problem:<br>*Gemma-2*: **1. Calculate the total investment:**<br>*Gemma-2*: * Josh spent $80,000 on the house + $50,000 on repairs = $130,000 total investment.<br>*LLaMA-3*: **2. Calculate the increased value of the house:**<br>*Rho-Math*: * 150% of $130,000 = 1.5 * 130,000 = 195,000. So the house is now worth 195,000.<br>*Gemma-2*: **3. Calculate the profit:**<br>*Gemma-2*: * The house is now worth $195,000 - $130,000 (investment) = $65,000 profit.<br>*LLaMA-3*: The answer is $65,000. |
| **BoE, Wrong answer** (*reward*: highest among candidates)<br><br>The original value of the house was $80,000. He put in an additional $50,000 in repairs. This increased the value of the house by 150%. So the new value of the house is $80,000 + 150% of $80,000 + $50,000. This is $80,000 + 1.5 * $80,000 + $50,000. This is $80,000 + $120,000 + $50,000. This is $250,000. So he made a profit of $250,000 - $130,000 = $120,000. |

Table 8: An example of the outputs generated by LE-MCTS and BoE to solve a test case in GSM8K. We highlight the first error made by the model in red.

```
Your task is to generate 16 math questions that match the same level of difficulty, required skills, and style as
the five given example problems. You then generate the step-by-step solutions and the final answer to each
generated question. Finally, format the generated outputs in this format: '{"question":"","answer":"","idx":}.

[Example 1] Janet's ducks lay 16 eggs per day. She eats three for breakfast every morning and bakes muffins
for her friends every day with four. She sells the remainder at the farmers' market daily for $2 per fresh duck
egg. How much in dollars does she make every day at the farmers' market?
[Example 2] A robe takes 2 bolts of blue fiber and half that much white fiber.  How many bolts in total does it
take?
[Example 3] 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?
[Example 4] James decides to run 3 sprints 3 times a week.  He runs 60 meters each sprint.  How many total
meters does he run a week?
[Example 5] Every day, Wendi feeds each of her chickens three cups of mixed chicken feed, containing seeds,
mealworms and vegetables to help keep them healthy.  She gives the chickens their feed in three separate
meals. In the morning, she gives her flock of chickens 15 cups of feed.  In the afternoon, she gives her
chickens another 25 cups of feed.  How many cups of feed does she need to give her chickens in the final
meal of the day if the size of Wendi's flock is 20 chickens?
```

Figure 5: A prompt for generating 16 synthetic examples analogous to those in GSM8K.

Your task is to generate 16 math questions that match the same level of difficulty, required skills, and style as the five given example problems. You then generate the step-by-step solutions and the final answer to each generated question. Finally, format the generated outputs in this format:
'{"problem": "", "solution": "", answer": "", "subject": "", "level": , "unique_id": ""}
, where “subject” is the topic of the question and “level” refers to the difficulty of the problem, ranging from 1 (grade-school level) to 5 (challenge level). Note that I want the questions with level 2 to 4.

[Example 1] Convert the point $(0,3)$ in rectangular coordinates to polar coordinates. Enter your answer in the form $(r,\\theta),$ where $r > 0$ and $0 \\le \\theta < 2 \\pi.$
[Example 2] Define\n\\[p = \\sum_{k = 1}^\\infty \\frac{1}{k^2} \\quad \\text{and} \\quad q = \\sum_{k = 1}^\\infty \\frac{1}{k^3}.\\]Find a way to write\n\\[\\sum_{j = 1}^\\infty \\sum_{k = 1}^\\infty \\frac{1}{(j + k)^3}\\]in terms of $p$ and $q.$
[Example 3] If $f(x) = \\frac{3x-2}{x-2}$, what is the value of $f(-2) +f(-1)+f(0)$? Express your answer as a common fraction.
[Example 4] How many positive whole-number divisors does 196 have?
[Example 5] The results of a cross-country team's training run are graphed below. Which student has the greatest average speed? [asy]\nfor ( int i = 1; i <= 7; ++i )\n{\n\ndraw((i,0)--(i,6));\n}\n\nfor ( int i = 1; i <= 5; ++i )\n{\n\ndraw((0,i)--(8,i));\n}\ndraw((-0.5,0)--(8,0), linewidth(1));\ndraw((0,-0.5)--(0,6), linewidth(1));\nlabel(\"$O$\", (0,0), SW);\nlabel(scale(.85)*rotate(90)*\"distance\", (0, 3), W);\nlabel(scale(.85)*\"time\", (4, 0), S);\ndot((1.25, 4.5));\nlabel(scale(.85)*\"Evelyn\", (1.25, 4.8), N);\ndot((2.5, 2.2));\nlabel(scale(.85)*\"Briana\", (2.5, 2.2), S);\ndot((4.25,5.2));\nlabel(scale(.85)*\"Carla\", (4.25, 5.2), SE);\ndot((5.6, 2.8));\nlabel(scale(.85)*\"Debra\", (5.6, 2.8), N);\ndot((6.8, 1.4));\nlabel(scale(.85)*\"Angela\", (6.8, 1.4), E);\n[/asy]

Figure 6: A prompt for generating 16 synthetic examples analogous to those in MATH.

Your task is to generate 16 math questions that match the same level of difficulty, required skills, and style as the five given example problems. You then generate the step-by-step solutions and the final answer to each generated question. Finally, format the generated outputs in this format:
'{"Answer":,"Question":,"Equation":,"Body":"","Type":"","ID":"","idx":0}’
, where Body is the question body and the question is the actual question, and type is the question such as "subtraction".

[Example 1] There are 87 oranges and 290 bananas in Philip's collection. If the bananas are organized into 2 groups and oranges are organized into 93 groups. How big is each group of bananas?
[Example 2] Marco and his dad went strawberry picking. Marco's dad's strawberries weighed 11 pounds. If together their strawberries weighed 30 pounds. How much did Marco's strawberries weigh?
[Example 3] Edward spent $ 6 to buy 2 books each book costing him the same amount of money. Now he has $ 12. How much did each book cost?
[Example 4] Frank was reading through his favorite book. The book had 3 chapters, each with the same number of pages. It has a total of 594 pages. It took Frank 607 days to finish the book. How many pages are in each chapter?
[Example 5] There were 78 dollars in Olivia's wallet. She spent 15 dollars at a supermarket. How much money does she have left?

Figure 7: A prompt for generating 16 synthetic examples analogous to those in SVAMP.

Your task is to generate 16 math questions that match the same level of difficulty, required skills, and style as the five given example problems. You then generate the step-by-step solutions and the final answer to each generated question. Finally, format the generated outputs in this format:
'{"body":"","question":"","solution_type":"","answer":"","formula":"","idx":}'
, where the "body" is the question body and "question" is the actual question.

[Example 1] Seven red apples and two green apples are in the basket. How many apples are in the basket?
[Example 2] Ellen has six more balls than Marin. Marin has nine balls. How many balls does Ellen have?
[Example 3] Janet has nine oranges and Sharon has seven oranges. How many oranges do Janet and Sharon have together?
[Example 4] Allan brought two balloons and Jake brought four balloons to the park. How many balloons did Allan and Jake have in the park?
[Example 5] Adam has five more apples than Jackie. Jackie has nine apples. How many apples does Adam have?

Figure 8: A prompt for generating 16 synthetic examples analogous to those in ASDiv.

Your task is to generate 16 math questions that match the same level of difficulty, required skills, and style as the five given example problems. You then generate the step-by-step solutions and the final answer to each generated question. Finally, format the generated outputs in this format:
'{"problem":"","rationale":"",,"options":"","correct":"","annotated_formula":"","linear_formula":"","type":""}'
, where the "correct" is the single character among {"a","b","c","d","e"}.

[Example 1] if log 8 x + log 8 1 \/ 6 = 1 \/ 3 , then the value of x is : a ) 12 , b ) 16 , c ) 18 , d ) 24 , e ) 26
[Example 2] the compound ratio of 5 : 6 , 3 : 2 and 6 : 5 ? a ) 1 : 1 , b ) 1 : 87 , c ) 1 : 6 , d ) 3 : 2 , e ) 1 : 2
[Example 3] two boys starts from the same place walking at the rate of 5 kmph and 5.5 kmph respectively in the same direction . what time will they take to be 8.5 km apart ? a ) 17 hr , b ) 14 hr , c ) 12 hr , d ) 19 hr , e ) 23 hr
[Example 4] set a of 8 positive integers may have the same element and have 40 . and set b of 8 positive integers must have different elements and have 40 . when m and n are the greatest possible differences between 40 and other elements \u2019 sums in set a and set b , respectively , m - n = ? a ) 21 , b ) 29 , c ) 23 , d ) 25 , e ) 26
[Example 5] a library has an average of 510 visitors on sundays and 240 on other day . the average number of visitors in a month of 10 days starting with sunday is a ) 280 , b ) 285 , c ) 290 , d ) 855 , e ) 275

Figure 9: A prompt for generating 16 synthetic examples analogous to those in MQA.

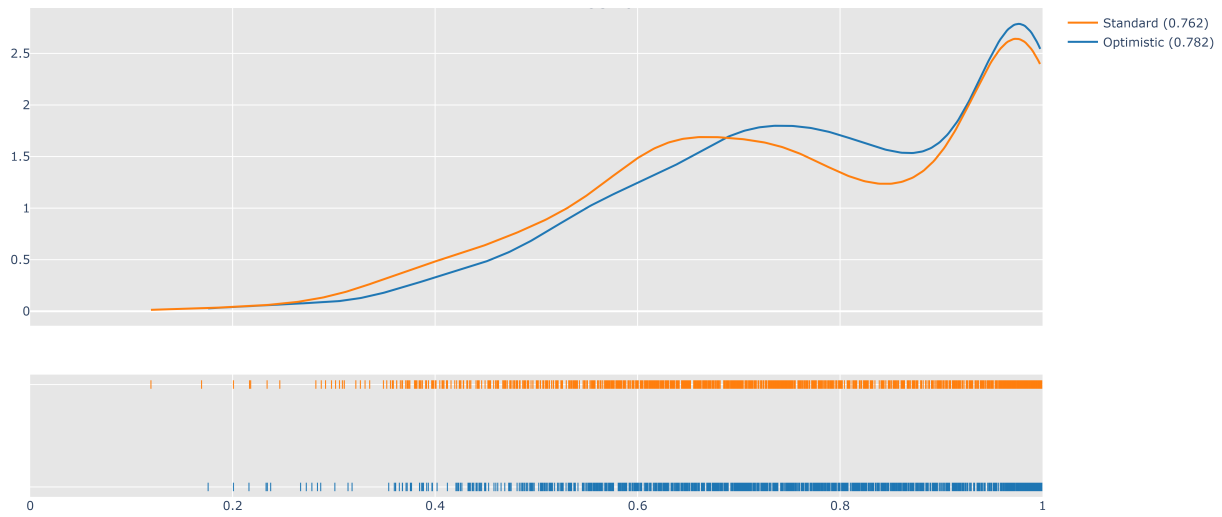


Figure 10: The distribution of the average reward for leaf nodes per example in GSM8K.

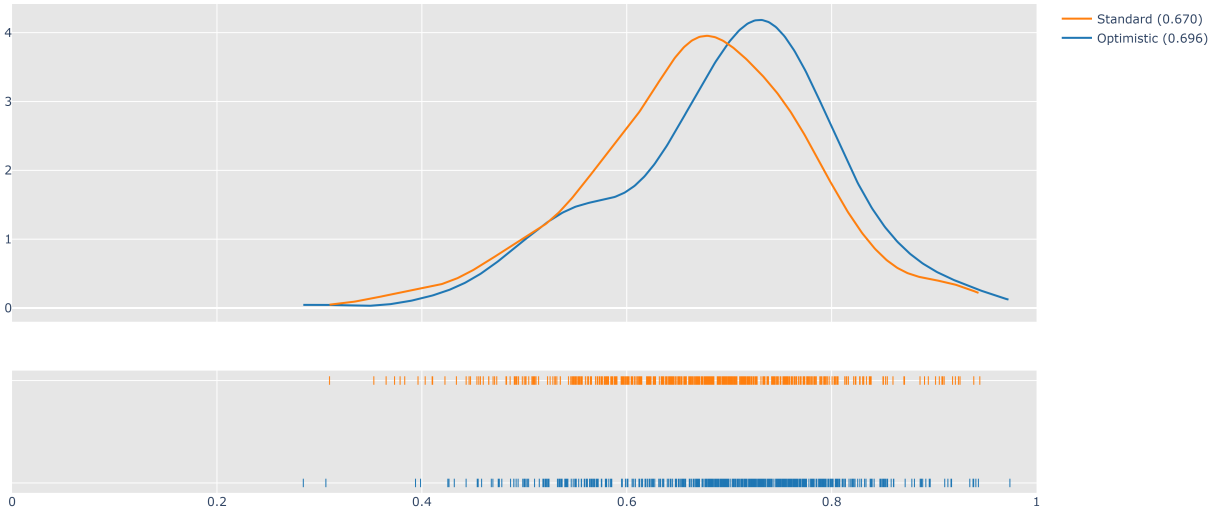


Figure 11: The distribution of the average reward for leaf nodes per example in MATH.

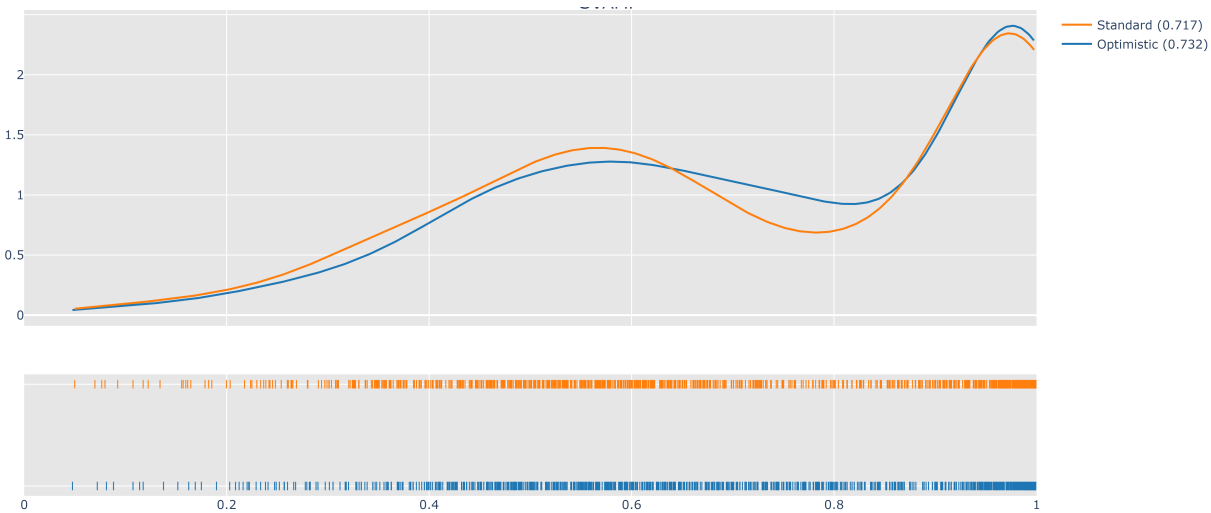


Figure 12: The distribution of the average reward for leaf nodes per example in SVAMP.

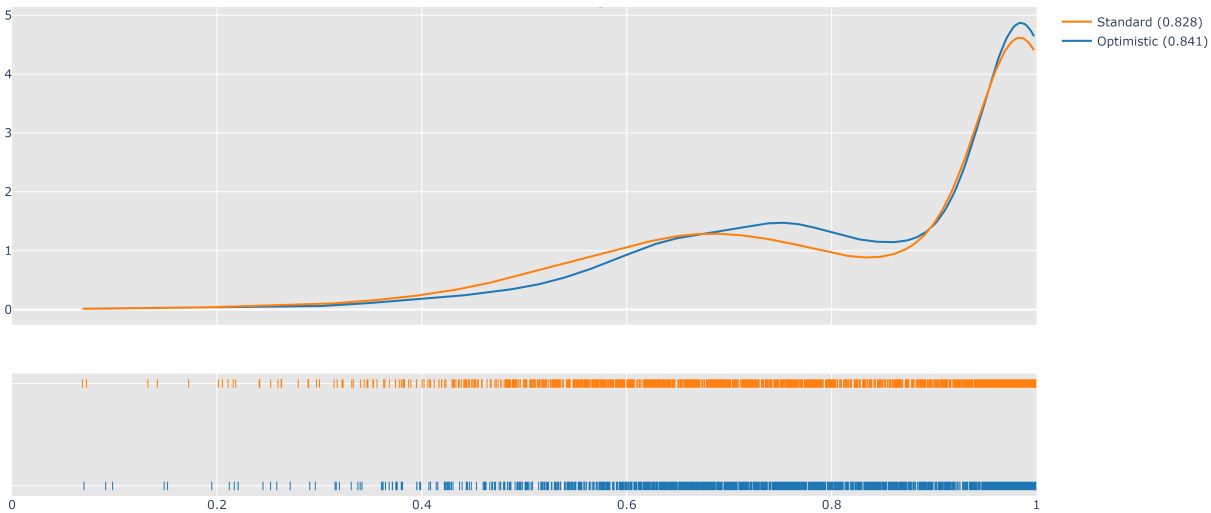


Figure 13: The distribution of the average reward for leaf nodes per example in ASDiv.

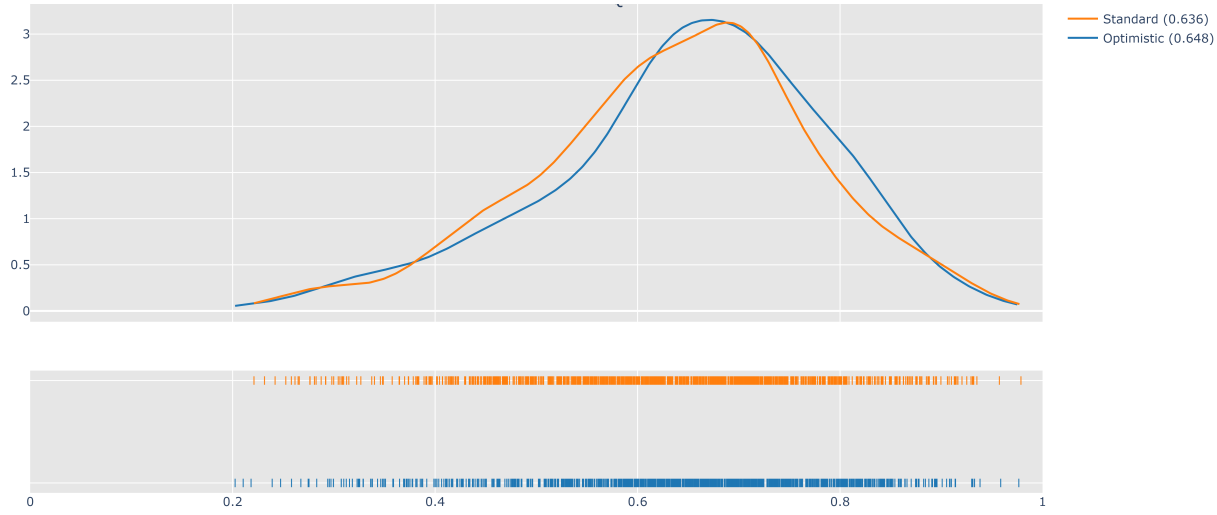


Figure 14: The distribution of the average reward for leaf nodes per example in MQA.

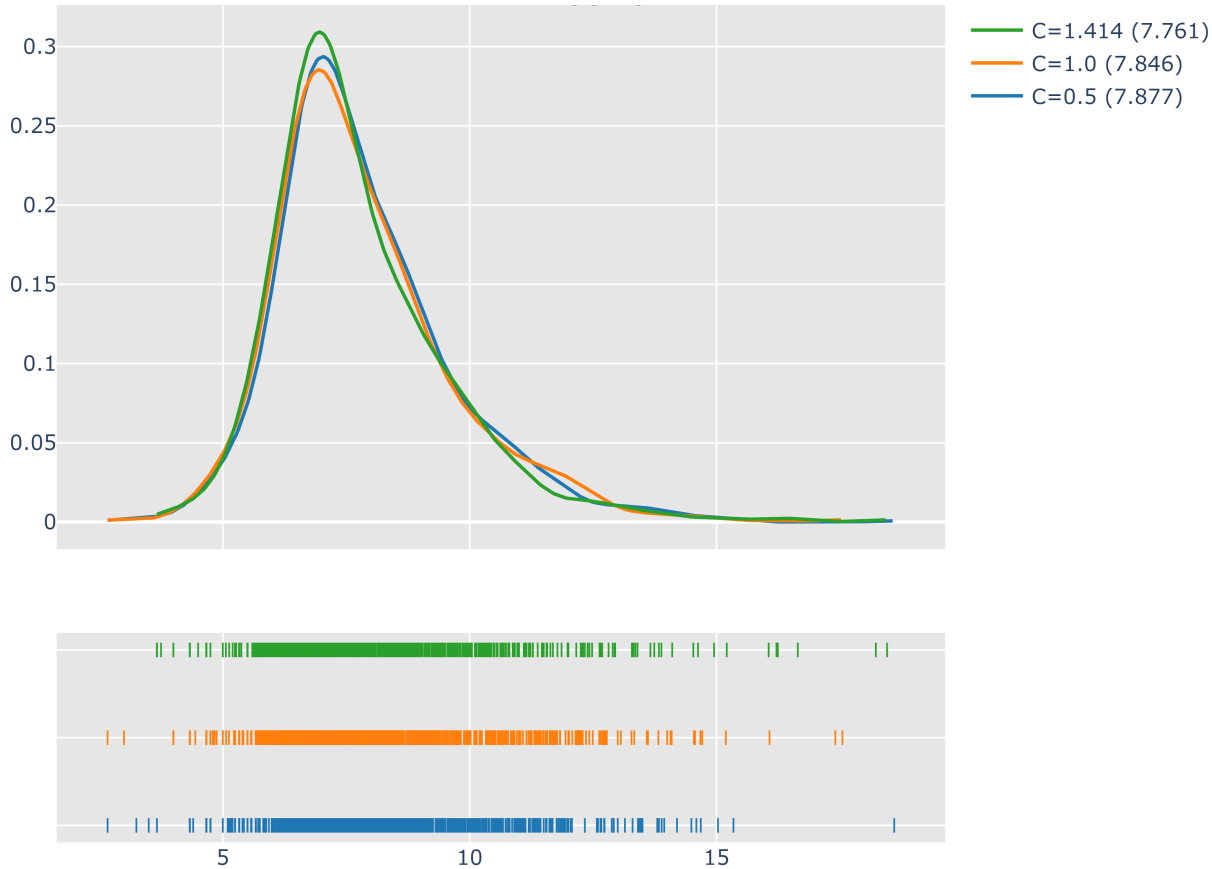


Figure 15: The distribution of the average depth of leaf nodes per example in GSM8K.

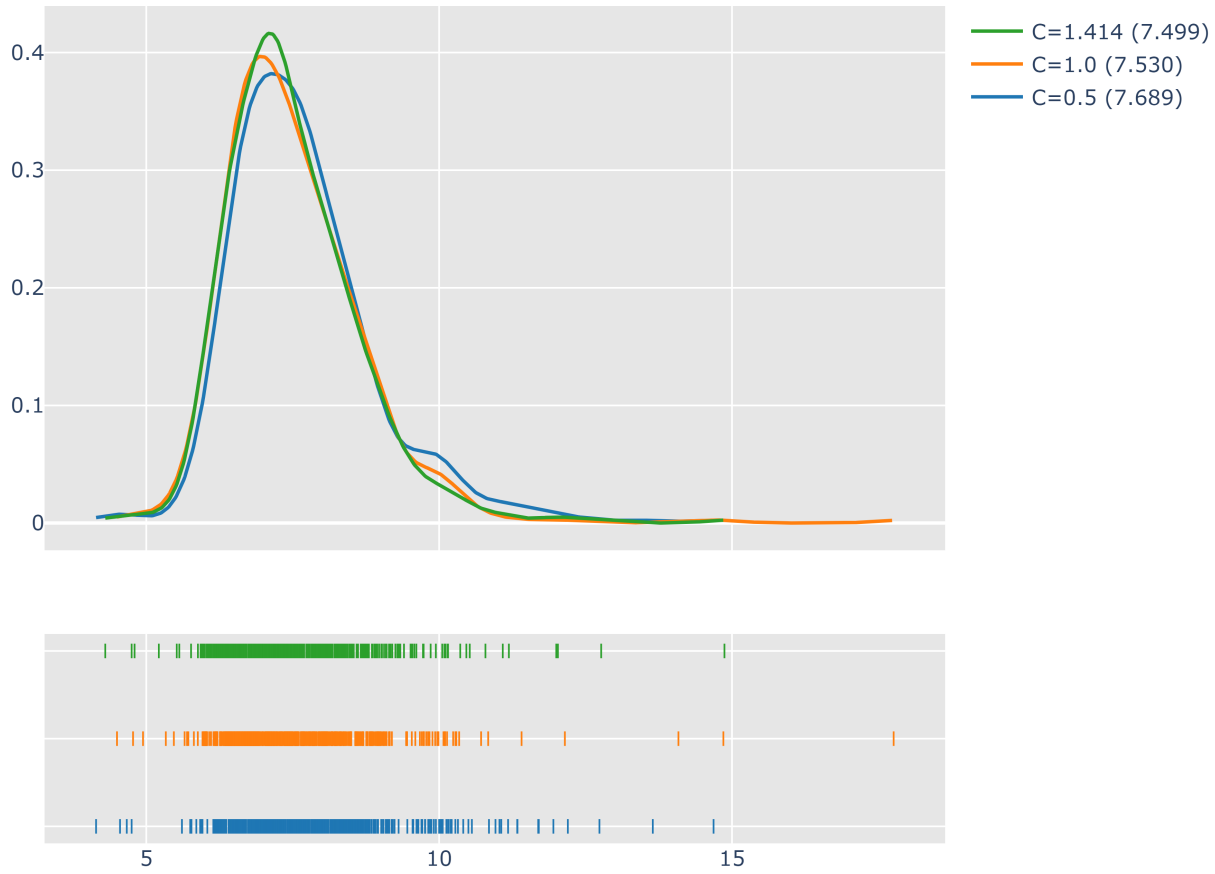


Figure 16: The distribution of the average depth of leaf nodes per example in MATH.

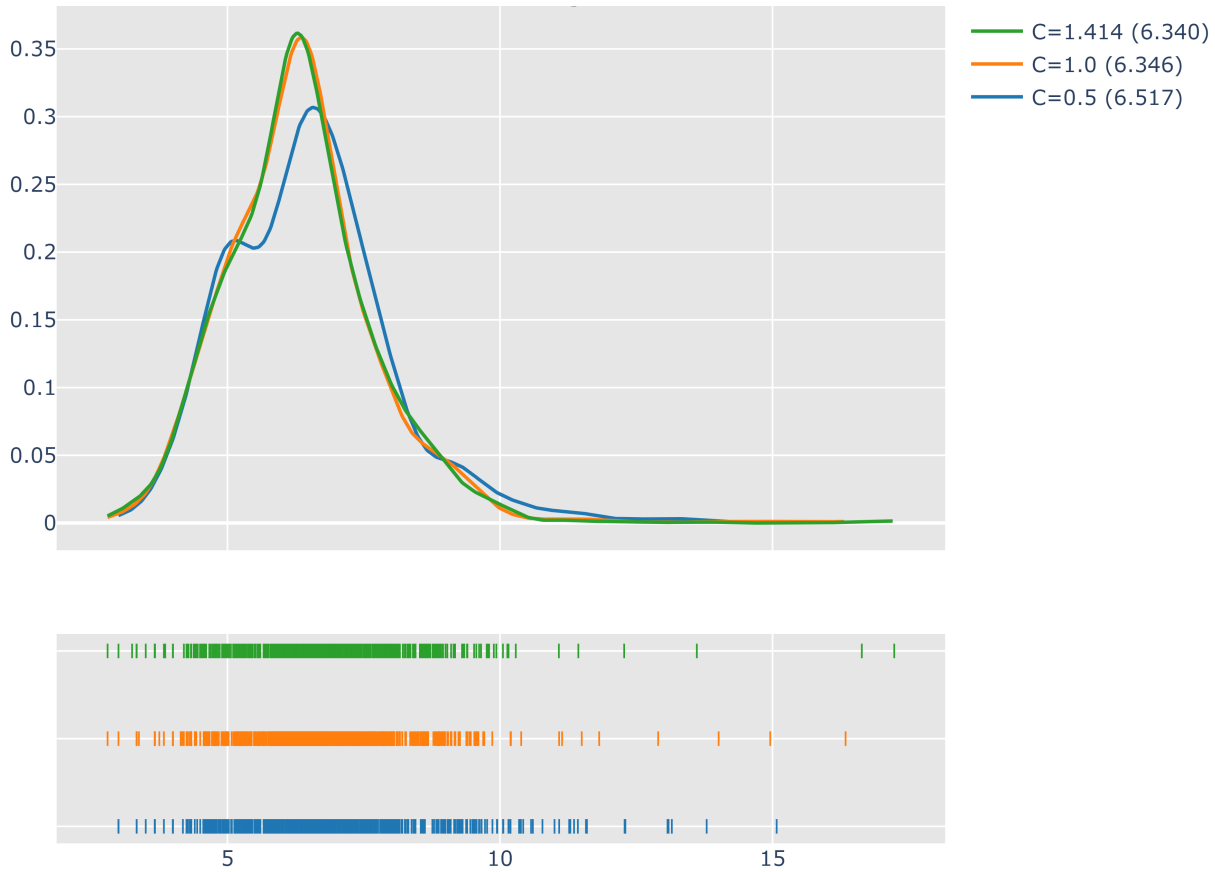


Figure 17: The distribution of the average depth of leaf nodes per example in SVAMP.

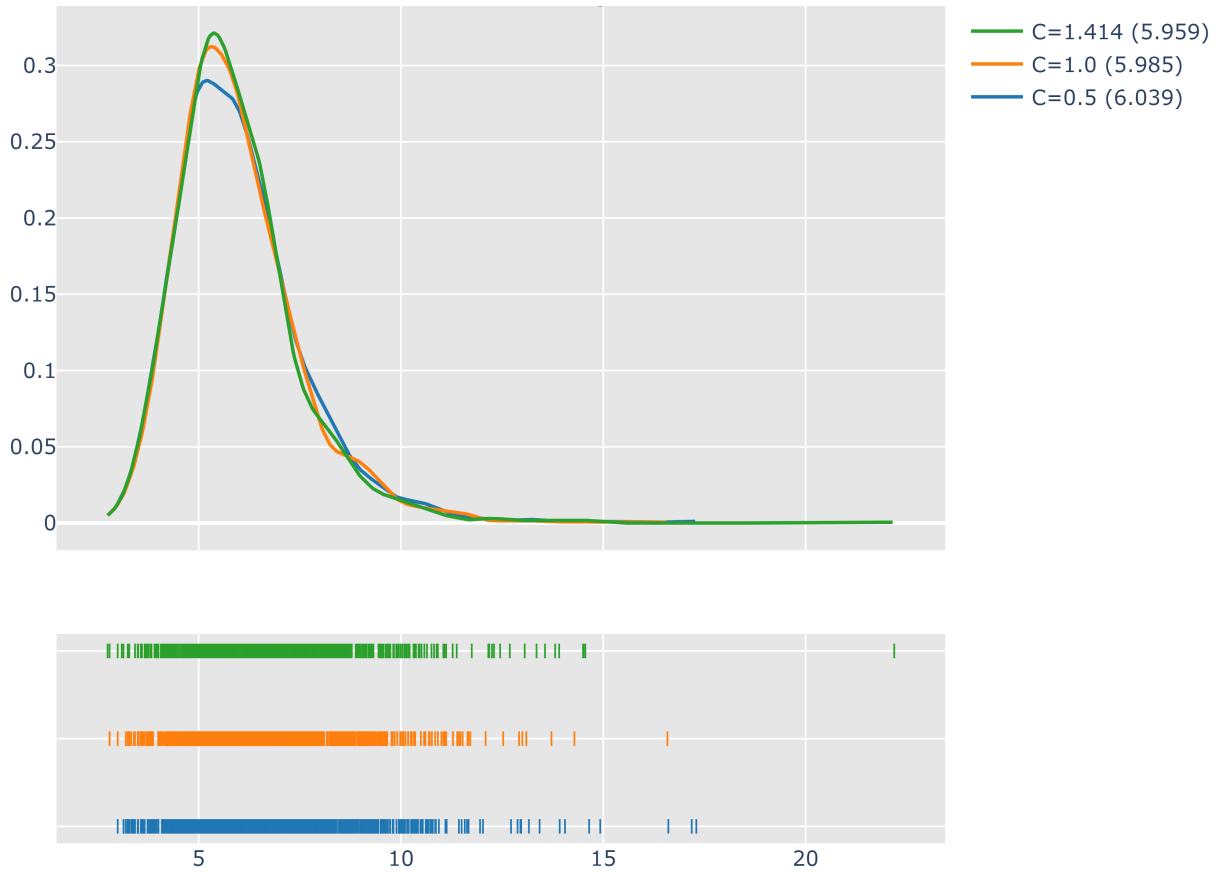


Figure 18: The distribution of the average depth of leaf nodes per example in ASDiv.

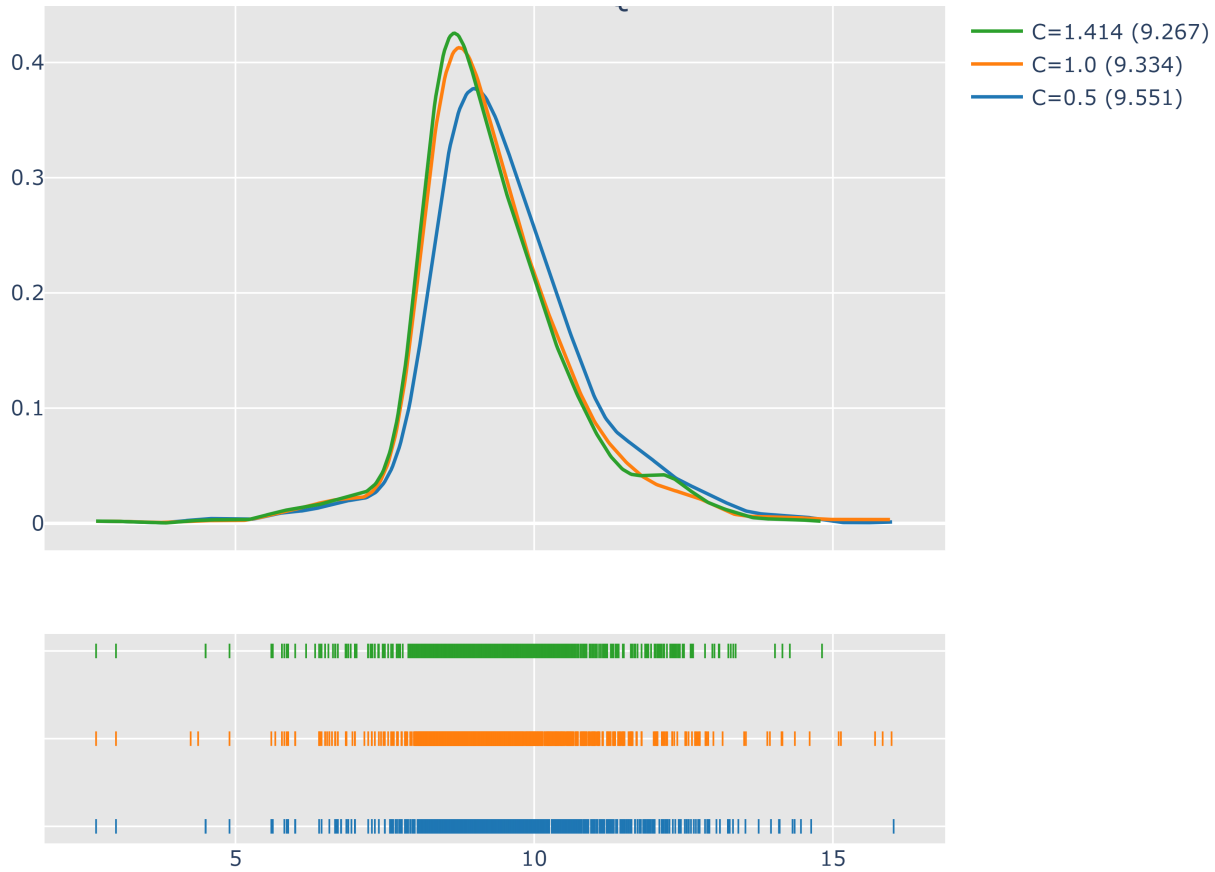


Figure 19: The distribution of the average depth of leaf nodes per example in MQA.