# FastMCTS: A Simple Sampling Strategy for Data Synthesis

**Peiji Li[1,2*], Kai Lv[1,2*], Yunfan Shao[1,2], Yichuan Ma[1,2], Linyang Li[2], Xiaoqing Zheng[1], Xipeng Qiu[1], Qipeng Guo[2†]**
[1]Fudan University
[2]Shanghai AI Laboratory
pjli24@m.fudan.edu.cn, {lilinyang, guoqipeng}@pjlab.org.cn

## Abstract

Synthetic high-quality multi-step reasoning data can significantly enhance the performance of large language models on various tasks. However, most existing methods rely on rejection sampling, which generates trajectories independently and suffers from inefficiency and imbalanced sampling across problems of varying difficulty. In this work, we introduce FastMCTS, an innovative data synthesis strategy inspired by Monte Carlo Tree Search. FastMCTS provides a more efficient sampling method for multi-step reasoning data, offering step-level evaluation signals and promoting balanced sampling across problems of different difficulty levels. Experiments on both English and Chinese reasoning datasets demonstrate that FastMCTS generates over 30% more correct reasoning paths compared to rejection sampling as the number of generated tokens scales up. Furthermore, under comparable synthetic data budgets, models trained on FastMCTS-generated data outperform those trained on rejection sampling data by 3.9% across multiple benchmarks. As a lightweight sampling strategy, FastMCTS offers a practical and efficient alternative for synthesizing high-quality reasoning data. Our code will be publicly released. [1]

## 1 Introduction

Large language models (LLMs) have achieved remarkable performance across various domains. Reasoning capability plays a crucial role in this success and serves as the foundation for further extending their application scope. For complex problems, LLMs typically require multi-step reasoning to arrive at final solutions. Synthesizing reasoning trajectories and using them for training has proven to be an effective approach to enhancing their reasoning capabilities.

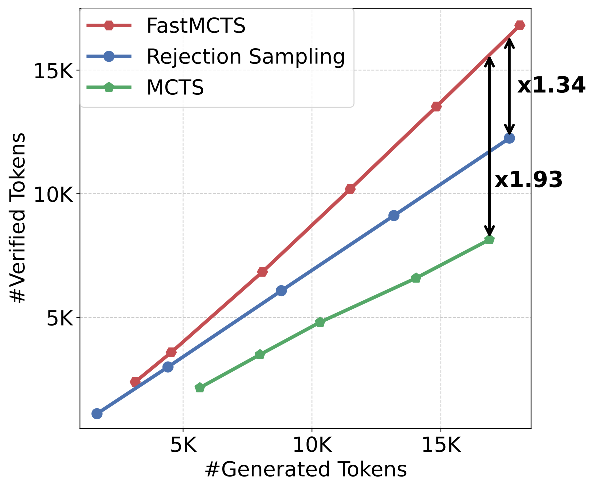



Figure 1: Comparison of generation efficiency of three sampling algorithms. "#Verified Tokens" represents the total tokens in all verified correct trajectories.

Currently, rejection sampling (Neal, 2003) is commonly used to synthesize correct trajectories for reasoning tasks. This approach generally involves generating multiple candidate responses through random sampling based on a given problem (Wei et al., 2022) , and then selecting the correct responses with the corresponding answers as synthetic training data. However, this random sampling method handles each attempt independently, constrained by the reasoning capacity of the policy model. As a result, it suffers from inefficiency particularly for long reasoning chains and complex problems, and it fails to provide step-level supervision during the synthesis process.

On the other hand, Monte Carlo Tree Search (MCTS) (Coulom, 2006), known for its ability to effectively explore state spaces, has been widely adopted in complex tasks such as board games. Some recent studies have also attempted to adapt MCTS for language models. However, the reasoning process of language models differs significantly

[*]Equal contribution
[†]Corresponding Author
[1]https://github.com/FlyingDutchman26/FastMCTS



*Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 24405–24422
July 27 - August 1, 2025 ©2025 Association for Computational Linguistics

from those of games like Go or chess. For instance, the state space in language model reasoning is often ill-defined, the computational cost is substantially higher, and the evaluation of reasoning outcomes tends to be more deterministic. As a result, directly applying MCTS to large-scale language generation tasks is less suitable.

In this work, we aim to efficiently deploy MCTS for data synthesis. We propose FastMCTS, an MCTS-inspired sample strategy for efficient data synthesis. To enhance data synthesis efficiency, we propose a dynamic balance mechanism between exploration and exploitation that adapts to problem complexity. Specifically, we introduce modifications to the selection phase of MCTS, enabling it to prioritize more valuable nodes rather than being limited to leaf nodes. Furthermore, vanilla MCTS employs a simulation process to evaluate node values. However, conducting complete sampling with LLMs is computationally expensive. To maximize the utility of tokens generated during the autoregressive decoding process of LLMs, we preserve each step of the complete reasoning trajectory generated during simulation as tree nodes, instead of discarding these reasoning steps after simulation. This do not influence the selection of the next most promising node in MCTS but serve as a caching mechanism to prevent redundant generation of reasoning trajectories. Figure 1 demonstrates the efficiency gains of FastMCTS compared to Rejection Sampling and vanilla MCTS in generating correct trajectory tokens on Chinese high school math data.

Experiments on a wide range of mathematical problems demonstrate the superior data synthesis efficiency of FastMCTS. Compared to vanilla rejection sampling, FastMCTS synthesizes more correct reasoning trajectories, produces more effective tokens, and solves a larger number of problems. This advantage is particularly pronounced for challenging problems, leading to more balanced synthesis across varying difficulty levels. Besides, under comparable generation budgets, models trained on FastMCTS-synthesized data outperform those trained on baseline methods across various benchmarks of different complexity.

Further analysis validates the effectiveness of the proposed components and shows that step-level pairwise data constructed through FastMCTS can further boost model performance through methods like step or branch level Direct Preference Optimization. As a lightweight data synthesis strategy, we believe FastMCTS offers a superior alternative to vanilla rejection sampling due to its higher efficiency and ability to provide step-level supervision for multi-step reasoning tasks.

## 2 Related Work

**Synthetic Data for Reasoning Tasks** Synthetic data has become a key resource for improving the reasoning capabilities of large language models. Several studies (Yu et al., 2024; Xu et al., 2024) focus on generating new problem sets by rephrasing or augmenting existing training data. Other works (Mukherjee et al., 2023; Li et al., 2024) leverage strong models, such as GPT-4 (Achiam et al., 2023), to distill high-quality reasoning data, enhancing the reasoning capabilities of smaller models; some of these approaches also utilize code executors to further improve performance (Yue et al., 2023; Wang et al., 2024a; Toshniwal et al., 2024). Additionally, methods like (Wang et al., 2024b; Luo et al., 2024; Wang et al., 2024d) focus on synthesizing multi-step reasoning data and provide step-level supervision without the need for human annotation.

**Sampling Strategies for Data Synthesis** Sampling strategies play a crucial role in enhancing the reasoning and generation capabilities of large language models. Many approaches improve reasoning performance by sampling multiple reasoning paths and selecting the most promising ones. For instance, Self-Consistency (Wang et al., 2023) generates diverse reasoning paths and selects the most consistent answer. Other works (Yuan et al., 2023; Toshniwal et al., 2024; Tong et al., 2024) use strategies like rejection sampling (Neal, 2003) to generates candidate outputs and filters them based on predefined criteria or a reward model.

**Tree Search in LLM** Tree-search strategies have been shown to be highly effective in enhancing the reasoning capacity of large language models, as the nodes of the tree can naturally represent reasoning steps in the chain-of-thought (CoT) (Wei et al., 2022). Several studies (Yao et al., 2024; Hao et al., 2023; Zhang et al., 2024b; Tian et al., 2024) have employed tree search during inference to guide multi-step reasoning. In another stream of research (Feng et al., 2023; Chen et al., 2024; Xie et al., 2024; Zhang et al., 2024a; Wang et al., 2024c), Monte-Carlo Tree Search is used to generate tree-structured data for training, constructing preference data pairs or providing process supervi-

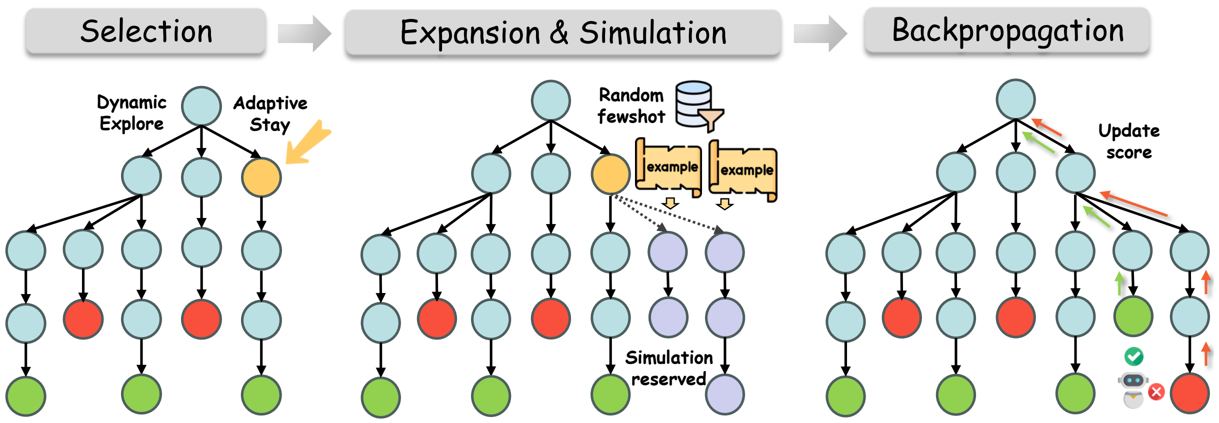



Figure 2: The overview of one iteration of FastMCTS

sion for CoT steps.

However, in synthetic data scenarios of LLMs, using MCTS can incur significant overhead due to simulation costs or rely on a trained process reward model for step supervision, leading to inefficiencies. To address these limitations, we propose FastMCTS, which efficiently synthesizes tree-structured multi-step reasoning data with high efficiency.

## 3 Preliminaries

**Rejection Sampling** Rejection sampling is a widely used synthetic-data method for obtaining high-quality data to enhance the reasoning capabilities of LLMs. Given an input question $q$, the process involves sampling multiple candidate responses $\{o^{(j)}\}_{j=1}^{N}$ from a language model. Each response $o^{(j)}$ is then evaluated based on predefined criteria, typically by comparing its final answer to a ground-truth solution using a rule-based function. Responses that pass this filtering step are considered correct and used to train the language model.

However, vanilla rejection sampling suffers from several limitations. For instance, the sampled data may exhibit imbalanced distributions (Tong et al., 2024). Moreover, due to the rule-based filtering mechanism, reasoning paths with errors in intermediate steps or those incorrectly discarded due to formatting issues are often excluded (Lightman et al., 2024). Our work addresses these issues effectively by introducing a more robust sampling strategy while achieving higher efficiency.

**Monte Carlo Tree Search** Monte Carlo Tree Search (MCTS) is a decision-making algorithm widely used in games like Go and complex decision processes (Silver et al., 2016, 2017). It builds a search tree through simulations to estimate the value of actions. In the context of language models, MCTS serves as a sampling strategy that can be combined with reward models to assist inference or synthesize multi-step reasoning data, providing step-level supervision for further training.

MCTS iteratively constructs a search tree through four phases: selection, expansion, simulation, and backpropagation (Browne et al., 2012). When applied to LLM inference, the input question $q$ is represented as the root node, and each reasoning step in the chain-of-thought (CoT) is represented as a child node. During selection, MCTS uses the Upper Confidence Bound for Trees (UCT) criterion to balance exploration and exploitation:

$$\mathrm{UCT}(i) = \frac{w_i}{n_i} + c \cdot \sqrt{\frac{\ln N_i}{n_i}} \tag{1}$$

where $n_i$ is the visit count for node $i$, $N_i$ is the visit count for its parent, $w_i$ is the cumulative value of descendant nodes, and $c$ is a hyperparameter.

Unlike board games, each roll-out in language models requires autoregressive inference, making the simulation process computationally expensive (Chen et al., 2024). The results of simulations are often discarded after backpropagation, further reducing sampling efficiency. As a result, directly applying MCTS for data synthesis incurs significant computational overhead.

## 4 Method

In our framework for synthetic data generation, for an input question $q$ and its solution with $T$ reasoning steps, the partial solution at time step $t$ is represented as state $s_t$, and the next reasoning step as action $a_{t+1}$. The language model is treated as a policy model $\pi_\theta$ and generates actions based

```
Algorithm 1: Selection phase of FastMCTS
Input: Current search tree T, difficulty thresholds l_high, l_low, UCT constant c
Output: Selected node in this iteration
▷ Recursively select node with Adaptive Stay Policy
current_node ← root
selected_node ← None
while selected_node is None do
    candidate_children ← current_node.children
    if number of candidate_children <= 1 or                         ▷ Adaptive Stay Policy
      all candidate_children are leaf nodes or
      current_node.visit_count > 1 and current_node.score ∈ (0, l_low] ∪ [l_high, 1) then
        selected_node ← current_node
        break
    if current_node.visit_count > 1 then
        c_current ← c · current_node.score                          ▷ Dynamic Exploration
    else
        c_current ← c
    candidate_node ← arg max_{node∈candidate_children} UCT(node, c_current)
    if candidate_node.visit_count > 1 and candidate_node.score <= l_low then
        selected_node ← candidate_node
    current_node ← candidate_node
```

on the current state and input question:

$$\pi_\theta(a_{t+1}|s_t) = \text{LLM}(a_{t+1}|s_t) \tag{2}$$

The transition to the next state is achieved by concatenating current state and next ction:

$$s_{t+1} = \text{Cat}(s_t, a_{t+1}) \tag{3}$$

where $s_t = (a_t, a_{t-1}, \ldots, a_1, q)$ represents the sequence of reasoning steps up to time $t$. We segment the reasoning trajectories into individual steps based on strings such as "Step 1", "Step 2", etc., with each step corresponding to a node in the tree structure. The details of how reasoning steps are separated are provided in Appendix A.

Our proposed method, FastMCTS, introduces several key improvements to vanilla MCTS algorithm, tailored for efficient and robust data synthesis in language models. In the following, we describe our algorithm in detail.

### 4.1 Selection with Adaptive Stay Policy

In the selection phase, Fast-MCTS recursively selects child nodes using the Upper Confidence Bound for Trees (UCT) criterion, as vanilla MCTS does, as defined in Equation 1. However, to improve efficiency and diversity, we introduce an Adaptive Stay Policy that dynamically adjusts the selection process based on the node's exploration status and estimated value.

In Adaptive Stay policy, selection does not necessarily proceed to leaf nodes as in vanilla MCTS. For states where the likelihood of being correct is either very high or very low, our method opts to "stay" rather than continuing selection. This approach prioritizes diversity for easier problems and attempts to explore at least one correct reasoning path for more challenging problems.

### 4.2 Dynamic Exploration

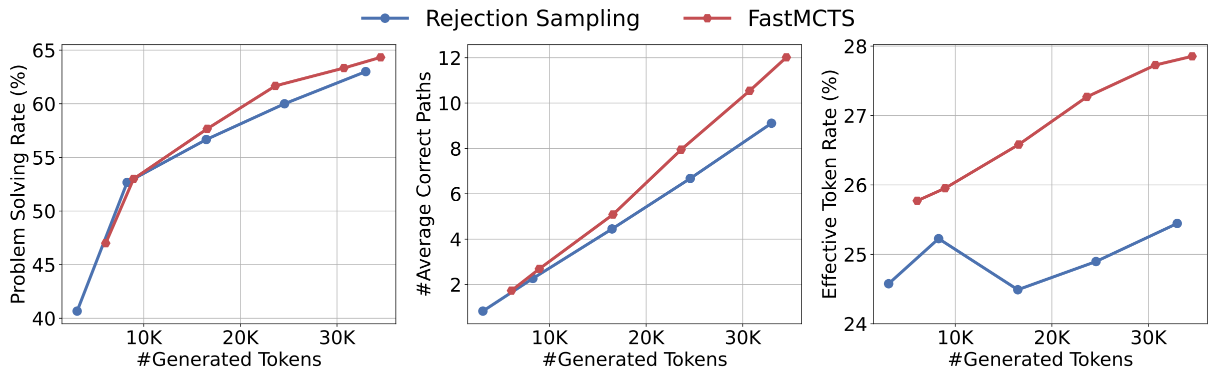
To enhance the search strategy, we dynamically adjust the parameter $c$ in UCT based on node scores. The score of one tree node is defined as the estimated value of taking an action (step), calculated by Monte Carlo Evaluation:

$$node.score = \frac{node.win\_count}{node.visit\_count} \tag{4}$$

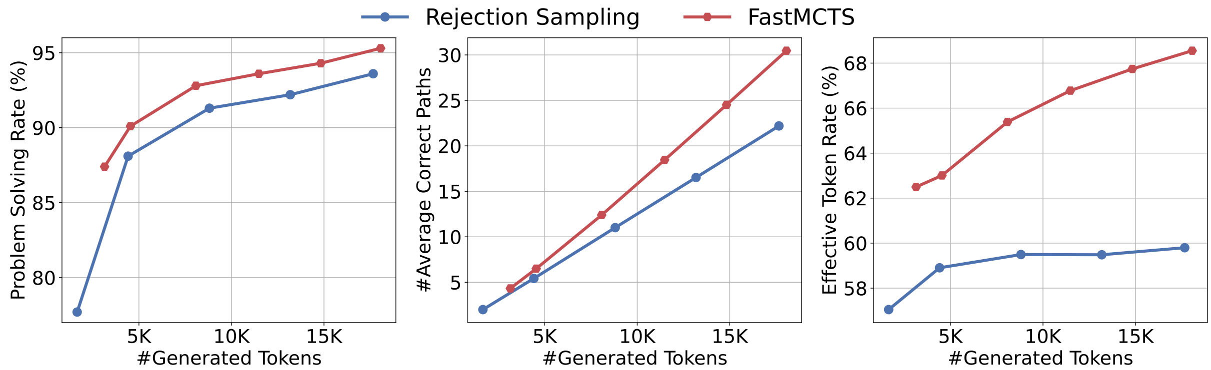
Then we adjuct $c$ by multiplying it with the node's score if the node has been visited more than once. This approach encourages exploration in promising states and prioritizes exploitation in less promising ones, aligning with the goal of data synthesis. The entire selection phase of the FastMCTS algorithm is demonstrated in Algorithm 1.

### 4.3 Reserve Simulation

Unlike board games like Go or chess, where the outcome of one random simulation does not necessarily reflect the quality of a specific state, LLM




(a) Sampling Efficiency on AIME




(b) Sampling Efficiency on CN High School Math

Figure 3: Comparison of sampling efficiency for FastMCTS and Rejection Sampling.

reasoning shows a strong correlation between the final answer and the correctness of the entire reasoning path. Therefore, simulation results in LLMs are valuable and should be preserved.

Inspired by this, we consolidate expansion and simulation into a single phase. Unlike vanilla MCTS, which discards simulation results, we preserve all newly generated paths as valid nodes and add them to our search tree. This significantly enhances sampling efficiency and integrates well with Adaptive Stay Policy. Since all trajectories are stored after selection, there is no need to delve deeply into leaf nodes during the search process.

### 4.4 Robustness Enhancements

To address variability in answer formats and logical errors in reasoning paths, we introduce a robustness enhancement mechanism. Instead of relying solely on rule-based answer matching, we use a LLM to evaluate the correctness of reasoning paths against the ground-truth answer. Additionally, we require the LLM to verify the correctness of intermediate steps within each path, aiming to identify logical errors and exclude trajectories that are guessed answers (e.g., multiple-choice questions). Details of our LLM evaluation methods are described in Appendix D.

Furthermore, to increase the diversity of generated reasoning paths, we prepend different random combinations of few-shot examples to each input string during simulation. To ensure a balanced distribution across mathematical disciplines, we constructed diverse exemplar sets for both Chinese and English datasets, covering domains such as trigonometry, analytic geometry, conic sections, derivatives, calculus, number theory, discrete mathematics, and linear algebra, ensuring sufficient diversity in prompt initialization. Each exemplar was standardized to enforce multi-step reasoning with explicit intermediate steps labeled as "Step 1", "Step 2", etc. This in-context learning approach promotes diverse reasoning paths, further enhancing data robustness.

### 4.5 Tree Construction and Data Utilization

The search tree is constructed iteratively, starting from the root node. The complete algorithm is outlined in Appendix B, and Figure 2 illustrates the flow of one iteration of FastMCTS.

We can construct training data from the tree structure. Specifically, correct reasoning paths are used for Supervised Fine-Tuning (SFT). Additionally, different branches within the tree nodes, based on their values, can be transformed into pair data for step-level and branch-level Direct Preference Optimization (Rafailov et al., 2023).

## 5 Experiment

### 5.1 Sampling Efficiency Comparison

In this section, we demonstrate the improvements in sampling efficiency of FastMCTS compared to Rejection Sampling. For our dataset, we utilized problems from the USA Mathematical Olympiad-level competition AIME up to the year 2023 (AI-MO, 2023a), along with Chinese high school mathematics problems collected from the internet, referred to as CN High School Math (Team, 2024). Specifically, we randomly selected 300 problems from AIME and 1000 problems from CN High School Math for our experiments. We then compared the efficiency of both methods in generating correct problem instances. We use the open-sourced LLM Qwen2.5-72B-Instruct (Yang et al., 2024a) and temperature is set to 1. Detailed generation settings are provided in Appendix E.

Our experimental results are shown in Figure 3. We gradually increased the number of generated tokens during sampling and compared three metrics for Rejection Sampling and FastMCTS. **Problem Solving Rate** refers to the average probability of generating at least one correct reasoning trajectories for a query. **Average Correct Paths** refers to the average number of correct reasoning trajectories generated for a query. **Effective Token Rate** refers to the proportion of generated tokens that belong to correct reasoning trajectories.

As shown in Figure 3, FastMCTS generates over 30% more correct reasoning paths compared to Rejection Sampling as the number of generated tokens scales up. Additionally, FastMCTS produces more effective tokens, demonstrating its efficiency in data synthesis. Furthermore, FastMCTS achieves a higher Problem Solving Rate than Rejection Sampling. This is because diverse few-shot examples are prepended as context for each expanded branch before simulation, enhancing the diversity of generated reasoning paths and increasing the likelihood of finding the correct solution.

| | Rejection Sampling | FastMCTS |
|---|---|---|
| | *EN Math Hard* | |
| # Tokens | 27.8K | 26.2K |
| # Trajectories | 3.46 | **5.88** |
| | *CN High School Math Hard* | |
| # Tokens | 18.2K | 17.4K |
| # Trajectories | 8.15 | **13.70** |

Table 1: Comparison of synthetic data generation costs between Rejection Sampling and FastMCTS under the experimental settings of Section 5.2. The row "# Tokens" indicates the average number of tokens generated per problem during the sampling phase. The row "# Trajectories" indicates the average number of correct reasoning paths acquired per problem.

### 5.2 Training Performance Comparison

#### 5.2.1 Experimental Setup

In addition to the comparison of sampling efficiency, we also evaluated the training performance on datasets generated using FastMCTS versus those generated using Rejection Sampling, with comparable computational budgets. To facilitate a more general comparison, we conducted experiments on datasets with two different distributions, specifically Chinese and English.

**Training Data Generation** For English data, we selected 46,000 problems from a wide range of math data including Numina-Math (LI et al., 2024), MetaMath (Yu et al., 2023), and the training set of InternLM-Math (Ying et al., 2024). For Chinese data, we selected 50,000 problems from Chinese high school math problems collected from the Internet(Team, 2024). We used heuristic strategies and model evaluations to filter out simpler problems, retaining multiple-choice, fill-in-the-blank, and solution-type questions while excluding proof and diagram-drawing problems. More details for our data selection process are provided in Appendix C. We refer to these two datasets after selection as *EN Math Hard* and *CN High School Math Hard*. We used Qwen2.5-72B-Instruct as the policy model and other sampling settings are described in Appendix E. To ensure a fair comparison, we controlled the computational costs of both sampling strategies to be comparable. The specific computational costs for both datasets are detailed in Table 1. Under this configuration, FastMCTS generates fewer tokens per query while acquiring more correct reasoning trajectories compared to rejection sampling.

| Method | #Data | Base Level | High School Level | | Competition Level | | | Olympiad Level | | Avg. |
|---|---|---|---|---|---|---|---|---|---|---|
| | | GSM8K | Gaokao Math | SAT Math | AIME24 | AMC23 | MATH | Olympiad Bench | OmniMath | |
| Qwen2.5-7B | - | 88.2 | 62.6 | 70.6 | 0 | 47.5 | 66.8 | 26.2 | 35.5 | 49.7 |
| *Training Trajectories per Problem $\leq$ 5* | | | | | | | | | | |
| RS | 111K | <u>89.1</u> | 62.6 | 70.6 | 6.7 | 52.5 | 72.0 | 27.6 | 38.3 | 52.4 |
| FastMCTS | 132K | 88.9 | <u>63.6</u> | <u>74.5</u> | <u>13.3</u> | <u>57.5</u> | <u>73.0</u> | <u>28.1</u> | **39.8** | <u>54.8</u> |
| *Training Trajectories per Problem $\leq$ 10* | | | | | | | | | | |
| RS | 167K | 89.4 | 62.6 | 72.6 | 6.7 | 50.0 | 70.8 | 26.3 | 37.5 | 52.0 |
| FastMCTS | 223K | **90.0** | <u>64.0</u> | <u>74.5</u> | <u>13.3</u> | <u>57.5</u> | <u>72.0</u> | <u>27.3</u> | <u>38.7</u> | <u>54.7</u> |
| *Training Trajectories per Problem $\leq$ 16* | | | | | | | | | | |
| RS | 197K | 87.1 | **65.1** | 72.6 | 10.0 | 52.5 | 70.0 | 27.1 | 37.2 | 52.7 |
| FastMCTS | 288K | 88.9 | 63.8 | 72.6 | **20.0** | **60.0** | 74.0 | 27.5 | 38.3 | 55.6 |
| + Branch-DPO | 152K | <u>89.9</u> | 65.0 | **76.5** | **20.0** | 57.5 | **75.4** | **29.6** | <u>39.2</u> | **56.6** |

Table 2: The results of model performance trained on EN Math Hard dataset synthesized by Rejection Sampling and FastMCTS with comparable generation cost. RS refers to synthetic dataset generated though rejection sampling. **Bold** indicates the best value, and <u>underlined</u> indicates the best value within a group.

| Method | #Data | Gaokao24 | CMATH |
|---|---|---|---|
| Qwen2.5-7B | - | 33.3 | 85.8 |
| *Training Trajectories per Problem $\leq$ 5* | | | |
| RS | 158K | 58.0 | 89.3 |
| FastMCTS | 198K | <u>59.4</u> | **90.8** |
| *Training Trajectories per Problem $\leq$ 10* | | | |
| RS | 250K | 59.4 | 89.3 |
| FastMCTS | 359K | <u>60.9</u> | <u>89.5</u> |
| *Training Trajectories per Problem $\leq$ 16* | | | |
| RS | 305K | 60.9 | 88.8 |
| FastMCTS | 502K | **62.3** | 89.3 |
| + Branch-DPO | 215K | **62.3** | <u>89.8</u> |

Table 3: The results of model performance trained on CN High School Math Hard dataset synthesized by Rejection Sampling and FastMCTS with comparable generation cost. RS refers to synthetic dataset generated though rejection sampling. **Bold** indicates the best value, and <u>underlined</u> indicates the best value within a group.

**Baselines** We use Qwen2.5-7B (Yang et al., 2024a) and compare its performance when trained on data synthesized by FastMCTS and Rejection Sampling. For both methods, synthesized data is constructed into supervised fine-tuning datasets by randomly sampling different maximum limits of correct trajectories. For FastMCTS, we additionally construct preference data from its tree structures, including step-level and branch-level pairs, which are used for a second-phase Branch-DPO training. Detailed data construction and training setups are provided in Appendix F and G.

#### 5.2.2 Main Results

We evaluated our models across a variety of mathematical benchmarks. All models are assessed in a zero-shot setting, employing greedy decoding for evaluation purposes.

For models trained on data synthesized from EN Math Hard, we evaluated on GSM8K (Cobbe et al., 2021) for baseline assessment, Gaokao Bench Math (Tang et al., 2024) and SAT-Math (Tang et al., 2024) for high school-level problems, AIME24 (AI-MO, 2024), AMC23 (AI-MO, 2023b), and MATH-500 (Hendrycks et al., 2021; Lightman et al., 2024) for competition-level challenges, and Olympiad Bench (He et al., 2024) and OmniMath (Gao et al., 2024) for olympiad-level tasks. For models trained on CN High School Math Hard, we evaluated on 69 text-only problems from the 2024 Chinese Gaokao(National Higher Education Entrance Examination) and CMATH (Wei et al., 2023) for foundational performance. Our training data are carefully curated to ensure no overlap with these evaluation benchmarks.

The training results are presented in Table 2 and Table 3. Key findings include:

1. Under comparable generation budgets, models trained on FastMCTS-sampled data consistently outperform those trained on rejection sampling data.

2. The performance of models trained on FastMCTS-generated data improves as the number of reasoning trajectories per problem increases, while models trained on rejection sampling data show limited and inconsistent improvement.

| Method | EN Math Hard | CN High School Math Hard |
|---|---|---|
| Rejection Sampling | 2.10 | 1.79 |
| FastMCTS | **2.23** | **2.10** |

Table 4: The entropy comparison of difficulty level distributions (see Figure 4) in data synthesis methods.

| Method | Solving Rate(%) | #Correct Path |
|---|---|---|
| Rejection Sampling | 61.3 | 7.22 |
| FastMCTS | **61.7** | **7.95** |
| w/o fewshot | 60.7 | 7.37 |
| w/o stay | 55.9 | 7.59 |
| w/o dynamic | **61.7** | 7.28 |
| w/o stay & dynamic | 55.9 | 7.32 |

Table 5: Ablation study

2. FastMCTS-generated data can be effectively reused for Branch-DPO training, further enhancing reasoning performance.

These results demonstrate that FastMCTS-synthesized data is more effective than rejection sampling, even with a comparable or lesser generation budget. For FastMCTS, model performance improves with an increase in the number of trajectories used for training, and additional gains can be achieved through DPO by utilizing step-level scores from tree-structured data.

To further validate the effectiveness and robustness of our methods, we also conducted experiments on models of different series with different parameter sizes. Results could be found in Appendix H.

## 5.3 Analysis

### 5.3.1 Difficulty-Aware Sampling in FastMCTS

As described in Section 4.1, FastMCTS dynamically adapts the search process according to the problem difficulty. This adaptation results in a more balanced distribution of problems across different difficulty levels. Consequently, the data generated by FastMCTS is not only larger in quantity but also more effective for training purposes.

To analyze this, we categorize problems from our dataset into five difficulty levels based on the probability of sampling a correct answer using rejection sampling. We then compare the number of correct trajectories generated by both FastMCTS and rejection sampling for each level.

The results in Figure 4 show that FastMCTS achieves a more balanced distribution across difficulty levels than rejection sampling, particularly for higher-difficulty problems. These results highlight FastMCTS's difficulty-aware feature. During tree search, as iterations increase, Monte Carlo-estimated scores become more accurate. For harder problems, FastMCTS tends to sample branches with higher success probabilities, while for easier problems, it degenerates to rejection sampling, mainly focusing on diversity.

In Table 4, we also report the entropy of the distribution presented in Figure 4, which serves as a quantitative measure of its uniformity. The data synthesized by FastMCTS exhibits a higher entropy value, indicating a more uniform distribution across difficulty tiers compared to Rejection Sampling.

These findings explain the effectiveness of data synthesized by FastMCTS. Although tree-search process may reduce diversity due to shared prefixes, FastMCTS achieves a more balanced distribution across problems of varying difficulty levels.

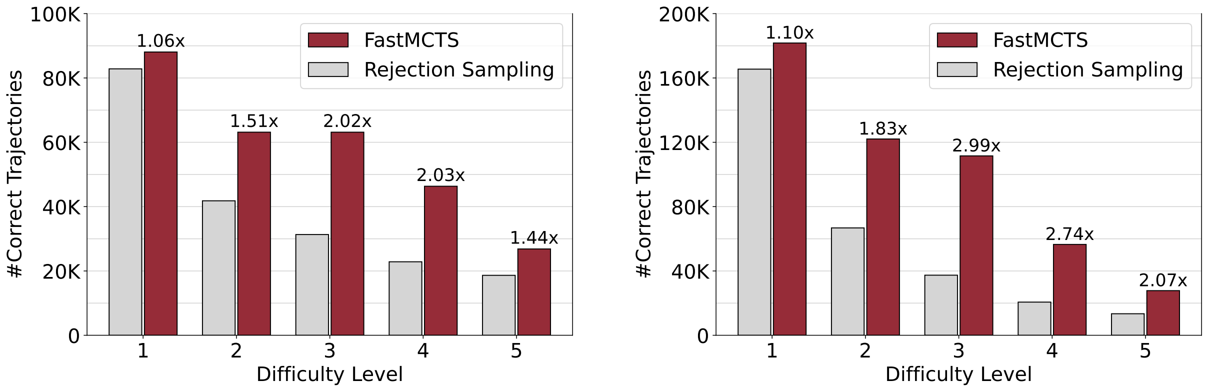
### 5.3.2 Ablation Study

For our ablation study, we compare the efficiency of FastMCTS with and without Adaptive Stay and Dynamic Exploration, using Rejection Sampling as the baseline. Experiments are conducted on 300 randomly selected AIME problems under the same settings provided in Appendix E. For each problem, we sample 25 trajectories: Rejection Sampling directly generates 25 trajectories, while FastMCTS performs 12 iterations of tree search with an initial degree of 3 and then expands 2 branches per phase, also yielding 25 trajectories.

From results in Table 5 (averaged over multiple runs), we could deduce that the Adaptive Stay policy primarily affects the problem solving rate. It decides whether to continue searching deeper or expand new branches based on the current node's score. As for Dynamic Exploration, it increases the efficiency of generating correct trajectories, as its absence reduces the average number of correct paths from 7.95 to 7.28. Removing few-shot examples leads to declines both in Problem Solving Rate and Average Correct Paths. These findings highlight the necessity and effectiveness of our proposed improvements in FastMCTS.

## 6 Conclusion

In this work, we introduce FastMCTS, an efficient sampling algorithm that leverages Monte Carlo Tree Search to synthesize high-quality multi-step




(a) Sampling Balance on EN Math Hard (b) Sampling Balance on CN High School Math Hard

Figure 4: Comparison of sampling balance across difficulty levels for Rejection Sampling and FastMCTS.

reasoning data for training large language models. Our approach not only improves the efficiency of data synthesis but also promotes a balanced sampling distribution across problems of varying difficulty, while providing step-level supervision for enhanced training like DPO. Experimental results demonstrate that FastMCTS outperforms rejection sampling in both sampling efficiency and training performance under comparable synthetic data budgets. We believe our method offers a practical solution for efficiently generating high-quality multi-step reasoning data and hope it inspires further research on data synthesis for language models.

## Limitations

Our work has several limitations. First, although we utilize a diverse range of data sources for data synthesis, our synthetic data is generated solely by the open-source model Qwen2.5-72B-Instruct for data generation. We do not employ stronger closed-source models like GPT-4 or models specifically fine-tuned for higher reasoning capabilities, such as Qwen-Math (Yang et al., 2024b), DeepSeek-R1 (DeepSeek-AI et al., 2025), or o1 (OpenAI et al., 2024). As a result, the performance of the trained models is not state-of-the-art.

Additionally, due to computational budget, we conduct our synthetic data experiments only in the math domain, we plan to extend our experiments to data from other domains in future work.

Finally, while FastMCTS-synthesized data achieve better training results due to its quantity and balanced distribution, the impact of prefix repetition in reasoning paths caused by the tree structure remains an open question, which we plan to investigate in future work.

## Acknowledgments

This work was supported by the National Key Research and Development Program of China (No. 2022ZD0160102). We express our sincere gratitude to Fudan University and Shanghai Artificial Intelligence Laboratory for providing an outstanding research environment, invaluable resources, and continuous support throughout this work.

## References

OpenAI Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haim ing Bao, Mo Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Benjamin Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Sim'on Posada Fishman, Juston Forte, Is abella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Raphael Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger

Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Lukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Hendrik Kirchner, Jamie Ryan Kiros, Matthew Knight, Daniel Kokotajlo, Lukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Ma teusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel P. Mossing, Tong Mu, Mira Murati, Oleg Murk, David M'ely, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Ouyang Long, Cullen O'Keefe, Jakub W. Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alexandre Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Pondé de Oliveira Pinto, Michael Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack W. Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario D. Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin D. Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas A. Tezak, Madeleine Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cer'on Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll L. Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qim ing Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. 2023. Gpt-4 technical report.

AI-MO. 2023a. Aime problems and solutions. https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions.

AI-MO. 2023b. American mathematics competitions. https://artofproblemsolving.com/wiki/index.php/AMC_12_Problems_and_Solutions.

AI-MO. 2024. Aime problems and solutions. https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions.

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

Guoxin Chen, Minpeng Liao, Chengxi Li, and Kai Fan. 2024. Alphamath almost zero: process supervision without process. *CoRR*, abs/2405.03553.

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. *CoRR*, abs/2110.14168.

Rémi Coulom. 2006. Efficient selectivity and backup operators in monte-carlo tree search. In *Computers and Games, 5th International Conference, CG 2006, Turin, Italy, May 29-31, 2006. Revised Papers*, volume 4630 of *Lecture Notes in Computer Science*, pages 72–83. Springer.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

Xidong Feng, Ziyu Wan, Muning Wen, Stephen Marcus McAleer, 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*.

Bofei Gao, Feifan Song, Zhe Yang, Zefan Cai, Yibo Miao, Qingxiu Dong, Lei Li, Chenghao Ma, Liang Chen, Runxin Xu, Zhengyang Tang, Benyou Wang, Daoguang Zan, Shanghaoran Quan, Ge Zhang, Lei Sha, Yichang Zhang, Xuancheng Ren, Tianyu Liu, and Baobao Chang. 2024. Omni-math: A universal olympiad level mathematic benchmark for large language models. *CoRR*, abs/2410.07985.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. 2023. Reasoning with language model is planning with world model. *arXiv preprint arXiv:2305.14992*.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. 2024. Olympiadbench: A challenging benchmark for promoting AGI with olympiad-level bilingual multimodal scientific problems. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 3828–3850. Association for Computational Linguistics.

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*.

Xin Lai, Zhuotao Tian, Yukang Chen, Senqiao Yang, Xiangru Peng, and Jiaya Jia. 2024. Step-dpo: Step-wise preference optimization for long-chain reasoning of llms. *Preprint*, arXiv:2406.18629.

Chen Li, Weiqi Wang, Jingcheng Hu, Yixuan Wei, Nanning Zheng, Han Hu, Zheng Zhang, and Houwen Peng. 2024. Common 7b language models already possess strong math capabilities. *CoRR*, abs/2403.04706.

Jia LI, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Costa Huang, Kashif Rasul, Longhui Yu, Albert Jiang, Ziju Shen, Zihan Qin, Bin Dong, Li Zhou, Yann Fleureau, Guillaume Lample, and Stanislas Polu. 2024. Numinamath. `[https://huggingface.co/AI-MO/NuminaMath-CoT](https://github.com/project-numina/aimo-progress-prize/blob/main/report/numina_dataset.pdf)`.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. Let's verify step by step. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Liangchen Luo, Yinxiao Liu, Rosanne Liu, Samrat Phatale, Harsh Lara, Yunxuan Li, Lei Shu, Yun Zhu, Lei Meng, Jiao Sun, and Abhinav Rastogi. 2024. Improve mathematical reasoning in language models by automated process supervision. *CoRR*, abs/2406.06592.

Subhabrata Mukherjee, Arindam Mitra, Ganesh Jawahar, Sahaj Agarwal, Hamid Palangi, and Ahmed Awadallah. 2023. Orca: Progressive learning from complex explanation traces of GPT-4. *CoRR*, abs/2306.02707.

Radford M Neal. 2003. Slice sampling. *The annals of statistics*, 31(3):705–767.

OpenAI, :, Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, Alex Iftimie, Alex Karpenko, Alex Tachard Passos, Alexander Neitz, Alexander Prokofiev, Alexander Wei, Allison Tam, Ally Bennett, Ananya Kumar, Andre Saraiva, Andrea Vallone, Andrew Duberstein, Andrew Kondrich, Andrey Mishchenko, Andy Applebaum, Angela Jiang, Ashvin Nair, Barret Zoph, Behrooz Ghorbani, Ben Rossen, Benjamin Sokolowsky, Boaz Barak, Bob McGrew, Borys Minaiev, Botao Hao, Bowen Baker, Brandon Houghton, Brandon McKinzie, Brydon Eastman, Camillo Lugaresi, Cary Bassin, Cary Hudson, Chak Ming Li, Charles de Bourcy, Chelsea Voss, Chen Shen, Chong Zhang, Chris Koch, Chris Orsinger, Christopher Hesse, Claudia Fischer, Clive Chan, Dan Roberts, Daniel Kappler, Daniel Levy, Daniel Selsam, David Dohan, David Farhi, David Mely, David Robinson, Dimitris Tsipras, Doug Li, Dragos Oprica, Eben Freeman, Eddie Zhang, Edmund Wong, Elizabeth Proehl, Enoch Cheung, Eric Mitchell, Eric Wallace, Erik Ritter, Evan Mays, Fan Wang, Felipe Petroski Such, Filippo Raso, Florencia Leoni, Foivos Tsimpourlas, Francis Song, Fred von Lohmann, Freddie Sulit, Geoff Salmon, Giambattista Parascandolo, Gildas Chabot, Grace Zhao, Greg Brockman, Guillaume

Leclerc, Hadi Salman, Haiming Bao, Hao Sheng, Hart Andrin, Hessam Bagherinezhad, Hongyu Ren, Hunter Lightman, Hyung Won Chung, Ian Kivlichan, Ian O'Connell, Ian Osband, Ignasi Clavera Gilaberte, Ilge Akkaya, Ilya Kostrikov, Ilya Sutskever, Irina Kofman, Jakub Pachocki, James Lennon, Jason Wei, Jean Harb, Jerry Twore, Jiacheng Feng, Jiahui Yu, Jiayi Weng, Jie Tang, Jieqi Yu, Joaquin Quiñonero Candela, Joe Palermo, Joel Parish, Johannes Heidecke, John Hallman, John Rizzo, Jonathan Gordon, Jonathan Uesato, Jonathan Ward, Joost Huizinga, Julie Wang, Kai Chen, Kai Xiao, Karan Singhal, Karina Nguyen, Karl Cobbe, Katy Shi, Kayla Wood, Kendra Rimbach, Keren Gu-Lemberg, Kevin Liu, Kevin Lu, Kevin Stone, Kevin Yu, Lama Ahmad, Lauren Yang, Leo Liu, Leon Maksin, Leyton Ho, Liam Fedus, Lilian Weng, Linden Li, Lindsay McCallum, Lindsey Held, Lorenz Kuhn, Lukas Kondraciuk, Lukasz Kaiser, Luke Metz, Madelaine Boyd, Maja Trebacz, Manas Joglekar, Mark Chen, Marko Tintor, Mason Meyer, Matt Jones, Matt Kaufer, Max Schwarzer, Meghan Shah, Mehmet Yatbaz, Melody Y. Guan, Mengyuan Xu, Mengyuan Yan, Mia Glaese, Mianna Chen, Michael Lampe, Michael Malek, Michele Wang, Michelle Fradin, Mike McClay, Mikhail Pavlov, Miles Wang, Mingxuan Wang, Mira Murati, Mo Bavarian, Mostafa Rohaninejad, Nat McAleese, Neil Chowdhury, Neil Chowdhury, Nick Ryder, Nikolas Tezak, Noam Brown, Ofir Nachum, Oleg Boiko, Oleg Murk, Olivia Watkins, Patrick Chao, Paul Ashbourne, Pavel Izmailov, Peter Zhokhov, Rachel Dias, Rahul Arora, Randall Lin, Rapha Gontijo Lopes, Raz Gaon, Reah Miyara, Reimar Leike, Renny Hwang, Rhythm Garg, Robin Brown, Roshan James, Rui Shu, Ryan Cheu, Ryan Greene, Saachi Jain, Sam Altman, Sam Toizer, Sam Toyer, Samuel Miserendino, Sandhini Agarwal, Santiago Hernandez, Sasha Baker, Scott McKinney, Scottie Yan, Shengjia Zhao, Shengli Hu, Shibani Santurkar, Shraman Ray Chaudhuri, Shuyuan Zhang, Siyuan Fu, Spencer Papay, Steph Lin, Suchir Balaji, Suvansh Sanjeev, Szymon Sidor, Tal Broda, Aidan Clark, Tao Wang, Taylor Gordon, Ted Sanders, Tejal Patwardhan, Thibault Sottiaux, Thomas Degry, Thomas Dimson, Tianhao Zheng, Timur Garipov, Tom Stasi, Trapit Bansal, Trevor Creech, Troy Peterson, Tyna Eloundou, Valerie Qi, Vineet Kosaraju, Vinnie Monaco, Vitchyr Pong, Vlad Fomenko, Weiyi Zheng, Wenda Zhou, Wes McCabe, Wojciech Zaremba, Yann Dubois, Yinghai Lu, Yining Chen, Young Cha, Yu Bai, Yuchen He, Yuchen Zhang, Yunyun Wang, Zheng Shao, and Zhuohan Li. 2024. Openai o1 system card. *Preprint*, arXiv:2412.16720.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D. Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. 2016. Mastering the game of go with deep neural networks and tree search. *Nat.*, 529(7587):484–489.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy P. Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. 2017. Mastering the game of go without human knowledge. *Nat.*, 550(7676):354–359.

Zhengyang Tang, Xingxing Zhang, Benyou Wang, and Furu Wei. 2024. Mathscale: Scaling instruction tuning for mathematical reasoning. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.

InternLM Team. 2024. https://github.com/internlm/internlm-wqx. `https://github.com/InternLM/InternLM-WQX`.

Ye Tian, Baolin Peng, Linfeng Song, Lifeng Jin, Dian Yu, Haitao Mi, and Dong Yu. 2024. Toward self-improvement of llms via imagination, searching, and criticizing. *arXiv preprint arXiv:2404.12253*.

Yuxuan Tong, Xiwen Zhang, Rui Wang, Ruidong Wu, and Junxian He. 2024. Dart-math: Difficulty-aware rejection tuning for mathematical problem-solving. *CoRR*, abs/2407.13690.

Shubham Toshniwal, Ivan Moshkov, Sean Narenthiran, Daria Gitman, Fei Jia, and Igor Gitman. 2024. Openmathinstruct-1: A 1.8 million math instruction tuning dataset. *CoRR*, abs/2402.10176.

Ke Wang, Houxing Ren, Aojun Zhou, Zimu Lu, Sichun Luo, Weikang Shi, Renrui Zhang, Linqi Song, Mingjie Zhan, and Hongsheng Li. 2024a. Mathcoder: Seamless code integration in llms for enhanced mathematical reasoning. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

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), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 9426–9439. Association for Computational Linguistics.

Xiyao Wang, Linfeng Song, Ye Tian, Dian Yu, Baolin Peng, Haitao Mi, Furong Huang, and Dong Yu. 2024c. Towards self-improvement of llms via mcts:

Leveraging stepwise knowledge with curriculum preference learning. *arXiv preprint arXiv:2410.06508*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Zihan Wang, Yunxuan Li, Yuexin Wu, Liangchen Luo, Le Hou, Hongkun Yu, and Jingbo Shang. 2024d. Multi-step problem solving through a verifier: An empirical analysis on model-induced process supervision. In *Findings of the Association for Computational Linguistics: EMNLP 2024, Miami, Florida, USA, November 12-16, 2024*, pages 7309–7319. Association for Computational Linguistics.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Tianwen Wei, Jian Luan, Wei Liu, Shuang Dong, and Bin Wang. 2023. Cmath: Can your language model pass chinese elementary school math test? *Preprint*, arXiv:2306.16636.

Yuxi Xie, Anirudh Goyal, Wenyue Zheng, Min-Yen Kan, Timothy P Lillicrap, Kenji Kawaguchi, and Michael Shieh. 2024. Monte carlo tree search boosts reasoning via iterative preference learning. *arXiv preprint arXiv:2405.00451*.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, Qingwei Lin, and Daxin Jiang. 2024. Wizardlm: Empowering large pre-trained language models to follow complex instructions. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. 2024a. Qwen2.5 technical report. *CoRR*, abs/2412.15115.

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

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.

Huaiyuan Ying, Shuo Zhang, Linyang Li, Zhejian Zhou, Yunfan Shao, Zhaoye Fei, Yichuan Ma, Jiawei Hong, Kuikun Liu, Ziyi Wang, Yudong Wang, Zijian Wu, Shuaibin Li, Fengzhe Zhou, Hongwei Liu, Songyang Zhang, Wenwei Zhang, Hang Yan, Xipeng Qiu, Jiayu Wang, Kai Chen, and Dahua Lin. 2024. Internlm-math: Open math large language models toward verifiable reasoning. *Preprint*, arXiv:2402.06332.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. 2023. Metamath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284*.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T. Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. 2024. Metamath: Bootstrap your own mathematical questions for large language models. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

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. *Preprint*, arXiv:2308.01825.

Xiang Yue, Xingwei Qu, Ge Zhang, Yao Fu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. 2023. Mammoth: Building math generalist models through hybrid instruction tuning. *CoRR*, abs/2309.05653.

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

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

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Jeff Huang, Chuyue Sun, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark W. Barrett, and Ying Sheng. 2023. Efficiently programming large language models using sglang. *CoRR*, abs/2312.07104.

## A Node Defination

In our application, each reasoning step is represented as a node. The reasoning trajectories are segmented into individual steps according to strings such as "Step 1", "Step 2", and so on, which are automatically generated by the model in its output. Each step may consist of a single sentence or span multiple sentences. Through the use of few-shot examples, the model learns to follow this structured format when generating multi-step reasoning paths, thereby enabling node-level partitioning. Below is one example of how each node is defined.

**Root(Query)**

Real numbers $x$ and $y$ with $x, y > 1$ satisfy $\log_x(y^x) = \log_y(x^{4y}) = 10$. What is the value of $xy$?
Please solve this task step by step and use the format like "Step 1:", "Step 2:" to structure your Chain of Thought (CoT) steps. Now, generate your solution starting from Step 1:

**Node 1**

Step 1: Start by analyzing the given equations. We have $\log_x(y^x) = 10$ and $\log_y(x^{4y}) = 10$.

**Node 2**

Step 2: Use the property of logarithms that $\log_b(a^c) = c\log_b(a)$ to simplify the equations. For the first equation, we have $\log_x(y^x) = x\log_x(y) = 10$. For the second equation, we have $\log_y(x^{4y}) = 4y\log_y(x) = 10$.

**Node 3**

Step 3: From Step 2, we can express $\log_x(y)$ and $\log_y(x)$ in terms of $x$ and $y$:

$$\log_x(y) = \frac{10}{x}$$

$$log_y(x) = \frac{10}{4y} = \frac{5}{2y}$$

**Node 4**

Step 4: Use the change of base formula $\log_a(b) = \frac{1}{\log_b(a)}$ to relate $\log_x(y)$ and $\log_y(x)$:

$$\log_x(y) = \frac{1}{\log_y(x)}$$

Substituting the values from Step 3, we get $\frac{10}{x} = \frac{1}{\frac{5}{2y}}$.

**Node 5**

Step 5: Simplify the equation from Step 4:

$$\frac{10}{x} = \frac{2y}{5}$$

Cross multiply to get $50 = 2xy$
Divide both sides by 2 to solve for $xy$: $xy = 25$.

**Node 6**

Step 6: Conclude that the value of $xy$ is 25. Thus, the answer is $\boxed{25}$.

## B FastMCTS Algorithms

The full FastMCTS algorithm is outlined in Algorithm 2.

## C Details of Training Data Selection

In our preliminary experiments to evaluate the efficiency of rejection sampling, we employed Qwen2.5-32B-Instruct to sample responses across our full Chinese and English dataset, with 5 samples generated per question. This allowed us to calculate a "pass rate" (percentage of correct solutions) for each question, which we used to stratify problem complexity. Questions with a 100% pass rate were deemed excessively simple and excluded from the dataset.

Additionally, we applied heuristic filtering strategies to remove ambiguous or low-quality problems:

- Rule-based exclusion: Problems containing keywords such as "proof", "prove", "show that", "find all" , or url/image extensions (e.g., "http", ".png", ".jpg", "www", ".svg", ".bmp") were automatically filtered out.

**Algorithm 2:** FastMCTS

**Input:** Input query $q$, ground truth $g$, few shot set $\mathcal{F}$, policy model $\pi_\theta$, verifier model $V_\phi$, initial degree $d_i$, expand degree $d_e$, iterations $N$, difficulty thresholds $l_{high}, l_{low}$, UCT constant $c$

**Output:** The search tree $T$ of input query $q$

**Initialize:** search tree $T$ with root $\leftarrow q$

```
while iter < N do
    ▷ Recursively select node with Adaptive Stay Policy
    current_node ← root
    selected_node ← None
    while selected_node is None do
        candidate_children ← current_node.children
        if number of candidate_children <= 1 or                              ▷ Adaptive Stay Policy
          all candidate_children are leaf nodes or
          current_node.visit_count > 1 and current_node.score ∈ (0, l_low] ∪ [l_high, 1) then
            selected_node ← current_node
            break
        if current_node.visit_count > 1 then
            c_current ← c · current_node.score                               ▷ Dynamic Exploration
        else
            c_current ← c
        candidate_node ← arg max_{node ∈ candidate_children} UCT(node, c_current)
        if candidate_node.visit_count > 1 and candidate_node.score <= l_low then
            selected_node ← candidate_node
        current_node ← candidate_node
    ▷ Expansion and Simulation
    Get current state s_t from root to selected_node: s_t = (a_t, a_{t-1}, ..., a_1, q)
    if candidate_node is root then
        Sample d_i partial trajectories {τ^(i)}_{i=1}^{d_i} ~ π_θ(τ | s_t, f^(i)),   f^(i) ⊆ F
                                                                             ▷ Sample with random fewshot
    else
        Sample d_e partial trajectories {τ^(i)}_{i=1}^{d_e} ~ π_θ(τ | s_t, f^(i)),   f^(i) ⊆ F
    Split {τ^(i)} to multi steps {(a^(i)_{t+1}, a^(i)_{t+2}, ..., a^(i)_{end})} and construct them as new branches of
      tree nodes {(node^(i)_{t+1}, node^(i)_{t+2}, ..., node^(i)_{end})}
    Append these new branches to selected_node                               ▷ Reserve all simulation results
    ▷ Backup
    Use verifier model V_φ to judge {τ^(i)}                                  ▷ Use LLM as verifer
    Backup score from newly expanded tree nodes using Monte Carlo Evaluation
```

| Model Name | AIME24 | MATH | GSM8K | AMC23 | Olympiad Bench | OmniMath | SAT Math | Gaokao Math | Avg. |
|---|---|---|---|---|---|---|---|---|---|
| Llama3.2_3b_base_RS | 0.0 | 31.0 | 55.5 | 15.0 | 10.7 | 10.7 | 50.2 | 30.0 | 25.4 |
| Llama3.2_3b_base_FastMCTS | 3.0 | 35.2 | 53.4 | 15.0 | 12.6 | 11.4 | 50.0 | 32.9 | **26.7** |
| Qwen2.5_3b_base_RS | 6.7 | 62.2 | 83.6 | 35.0 | 27.0 | 20.8 | 70.6 | 56.1 | 45.3 |
| Qwen2.5_3b_base_FastMCTS | 10.0 | 62.2 | 83.3 | 45.0 | 29.5 | 21.5 | 71.5 | 56.1 | **47.4** |
| Qwen2.5_7b_base_RS | 6.7 | 72.0 | 89.1 | 52.5 | 27.6 | 38.3 | 70.6 | 62.6 | 52.4 |
| Qwen2.5_7b_base_FastMCTS | 13.3 | 73.0 | 88.9 | 57.5 | 28.1 | 39.8 | 74.5 | 63.6 | **54.8** |

Table 6: The results of different model performance when trained with data generated by Rejection Sampling and FastMCTS.

- Format checks : Questions with formatting errors (e.g., broken LaTeX, incomplete sentences) were discarded.
- Deduplication : We removed duplicate entries via hash-based matching and ensured no overlap with the test set.

## D Details of Model Evaluation

As we have mentioned in Section 4.4, we propose to employ an LLM to verify the correctness of each reasoning path, aiming to identify logical errors and exclude trajectories that are guessed answers. For prompt design of LLM judge, an example prompt template is demonstrated in Figure 5.

Meanwhile, to reduce computational costs, we limited the maximum output length to 32 tokens (as only final answers are required). To ensure accuracy, we employed a majority voting strategy: the judge model verifies each answer N=3 times , and only consistent results across all trials are accepted. If inconsistencies arise, the verification is repeated until consensus is reached. This approach minimizes errors and outperforms rule-based matching in identifying nuanced correct answers.

The rigorous validation was critical because the synthesized data is used not only for supervised fine-tuning but also for Branch-DPO, where precise step-level evaluation (distinguishing true positives/negatives) is essential for Preference Optimization. All described details have been fully implemented in our code.

## E Sampling Settings

For all our sampling settings, we use SGLang (Zheng et al., 2023) as our inference engine and employ sampling generation with a temperature setting of 1 to ensure diversity. In FastMCTS, the constant $c$ in the UCT score is set to its default value of 1.414. Additionally, we utilize Qwen2.5-72B-Instruct as a LLM judger to verify the solutions.

We use an asynchronous approach in our implementation, allowing different branches of the search tree to be processed concurrently. Although FastMCTS requires multiple iterations to construct a search tree for each problem, this parallel processing allows us to perform inference on a large number of inputs simultaneously, thereby ensuring high efficiency.

In section 5.1, to scale up the sampling computation, for FastMCTS, we incrementally increased the number of iterations from 4 to 20, and the expansion degree (i.e., the number of nodes expanded after the selection phase) is varied from 1 to 2. For Rejection Sampling, we expanded the number of generated trajectories per query from 3 to 32.

In section 5.2, to obtain comparable sampling computation, for each query in the original dataset, we sampled multiple times (30 for English data and 24 for Chinese data) using rejection sampling. For FastMCTS, it starts with an initial degree of 3 at the root, expands by adding 2 branches in each expansion phase, and performs 16 iterations of tree search.

## F Training Data Construction

**Supervised Fine-tuning** After the sampling process, each problem is sampled with varying numbers of solution candidates. To investigate the impact of both training data size and the number of reasoning trajectories per problem, we impose constraints on the maximum number of solutions utilized per problem during the training process. This approach also helps maintain a balance between different problems.

For Rejection Sampling, we selecte correct trajectories for each problem randomly. For FastMCTS, our strategy involves prioritizing the selection of correct trajectories from various branches of the search tree. By doing so, we aim to maximize the

```
##Question##
{question}

##Student's Answer##
{model_output}

The standard answer for this question is as follows:
##Standard Answer##
{answer}

Now, based on the standard answer, determine whether the student's answer is correct.
(Please note that the same mathematical expression may have different formats or equivalent forms).
You only need to focus on:
1. Whether the student's answer matches the result of the standard answer.
2. Whether the student's answer seems to be guessed or is a vague answer. If the student's answer
is correct (if there are multiple questions, all sub-questions must be answered correctly),
please reply directly with:
**Correct Answer**
If the student's answer is incorrect, please reply directly with:
**Incorrect Answer**
```

Figure 5: Example of the Prompt Template Used for Model Evaluation

diversity of the training data.

**Branch-DPO** In addition to improving the efficiency of sampling correct reasoning paths, FastMCTS also provides step-level supervision information. Unlike rejection sampling, which generates multiple completely independent trajectories for each problem, FastMCTS constructs a search tree for each problem, where each node stores a score computed through Monte Carlo evaluation. This allows for step-level or branch-level preference optimization based on the scores of tree nodes.

Direct Preference Optimization (DPO) (Rafailov et al., 2023) has been widely adopted for model optimization due to its efficiency in utilizing pairwise preference data. It has also been applied to step-level preference optimization, as most undesirable trajectories do not initially contain errors (Lai et al., 2024; Xie et al., 2024; Chen et al., 2024; Wang et al., 2024c).

We propose a simple algorithm to construct preference data from the tree structures generated by FastMCTS. Our approach is based on the following assumptions:

1. For a multi-step reasoning trajectory, if the final result is correct and clear, all intermediate steps are considered correct.

2. If the final result is incorrect, the intermediate steps are not necessarily incorrect.

However, if a step has been simulated multiple times and its Monte Carlo-estimated score remains zero, it can be considered a **"low-quality node."** Based on this, we construct step-level or branch-level preference data. For any node in the tree, we examine its child nodes. If a child node is identified as low-quality, we construct step-level preference data between this node and a high-quality node that has led to correct results. If the child branches contain both correct and incorrect results but have only been simulated once, we cannot definitively assess the quality of individual steps and instead construct branch-level preference data.

In our experiments, for each search tree associated with one problem, we construct up to 5 step-level or branch-level preference pairs, resulting in an additional 152K(on CN High School Math Hard) and 215K(on En Math Hard) preference data points for DPO training. This approach further leverages the tree-structured data generated by FastMCTS.

## G Training Setups

We use Qwen2.5-7B as our base model and perform training on datasets generated by both FastMCTS and rejection sampling. For supervised fine-tuning, the maximum sequence length is set to 4096 tokens, and the global batch size is set to 32. We employ the Adam optimizer with a learning rate of 1e-5 and a linear warmup schedule with a warmup step ratio of 0.1. For all synthetic datasets, we train the model for 3 epochs and select the best checkpoint based on validation performance.

After supervised fine-tuning, we further refine the best checkpoint trained on FastMCTS-generated data using Branch-DPO for 3 epochs. The global batch size for Branch-DPO is set to

16, and the learning rate is set to 1e-6. The hyperparameter $\beta$ is set to 0.4. We use the AdamW optimizer with a cosine learning rate scheduler and a warmup ratio of 0.1.

## H Train with Different Models

We also evaluated our methods on LLMs of different series and sizes. Using the same experimental setup as in Section 5.2, we evaluated the En Math Hard dataset with less than 5 reasoning trajectories per problem. We compared the fine-tuning results(best checkpoint in 3 epochs) of synthetic data generated by FastMCTS and Rejection Sampling using two additional base models: Llama-3.2-3b-base and Qwen2.5-3b-base. The results are shown in Table 6:

These results demonstrate that even with different base models, under the same synthetic data cost, fine-tuning with data generated by FastMCTS consistently outperforms Rejection Sampling.

## I Performance against Recent Works

Our work proposes an algorithm designed to improve the efficiency and quality of synthetic data sampling for reasoning paths. Regarding the original data we selected, we primarily leverage open-source dataset and problems collected from website, this may not be the optimal instruction data for training SOTA models. However, we also compare our model performance compared to recent works with comparable model size, which is described in Table 7.

These results suggest that our model achieves competitive performance. We recognize that incorporating higher-quality dataset curation could further improve outcomes, which is a direction we plan to explore in future work.

| Model Name | AIME24 | MATH | GSM8K | AMC23 |
|---|---|---|---|---|
| GPT-4o | 9.3 | 76.6 | 92.9 | 47.5 |
| NuminaMath-CoT-7B | 0 | 55.8 | 76.3 | 27.5 |
| NuminaMath-TIR-7B | 16.7 | 68.1 | 84.6 | 50.0 |
| OpenMath2-Llama3.1-8B | 10.0 | 67.8 | 91.7 | 40.0 |
| rStarMath Policy 7B | 26.7 | 78.4 | 89.7 | 47.5 |
| FastMCTS 7B | 20.0 | 75.4 | 89.9 | 57.5 |

Table 7: Performance comparison with recent works on various benchmarks.