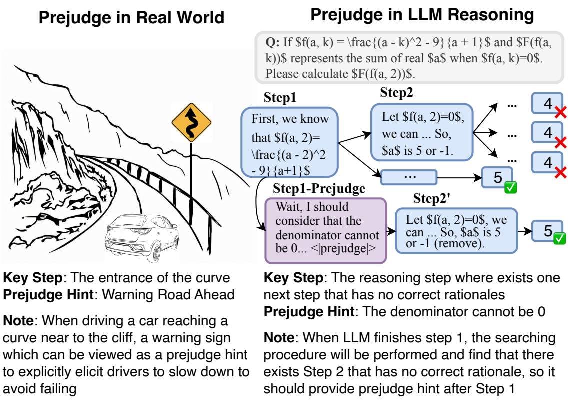
Prejudge-Before-Think: Enhancing Large Language Models at Test-Time by Process Prejudge Reasoning

Anonymous ACL submission

Abstract

In this paper, we introduce a new *process prejudge* strategy in LLM reasoning to demonstrate that bootstrapping with process prejudge allows the LLM to adaptively anticipate the errors encountered when advancing the subsequent reasoning steps, similar to people sometimes pausing to think about what mistakes may occur and how to avoid them, rather than relying solely on trial and error. Specifically, we define a prejudge node in the rationale, which represents a reasoning step, with at least one step that follows the prejudge node that has no paths toward the correct answer. To synthesize the prejudge reasoning process, we present an automated reasoning framework with a dynamic tree-searching strategy. This framework requires only one LLM to perform answer judging, response critiquing, prejudge generation, and thought completion. Furthermore, we develop a two-phase training mechanism with supervised fine-tuning (SFT) and reinforcement learning (RL) to further enhance the reasoning capabilities of LLMs. Experimental results from competition-level complex reasoning demonstrate that our method can teach the model to prejudge before thinking and significantly enhance the reasoning ability of LLMs [1].


Prejudge in Real World
Prejudge in LLM Reasoning
Q: If $f(a, k) = \frac{(a - k)^2 - 9}{a + 1}$ and $F(f(a, k))$ represents the sum of real $a$ when $f(a, k)=0$. Please calculate $F(f(a, 2))$.
Step1
First, we know that $f(a, 2)= \frac{(a - 2)^2 - 9}{a+1}$
Step2
Let $f(a, 2)=0$, we can ... So, $a$ is 5 or -1.
...
4
4
4
...
5
Step1-Prejudge
Wait, I should consider that the denominator cannot be 0... <|prejudge|>
Step2'
Let $f(a, 2)=0$, we can ... So, $a$ is 5 or -1 (remove).
5

Key Step: The entrance of the curve
Prejudge Hint: Warning Road Ahead

Note: When driving a car reaching a curve near to the cliff, a warning sign which can be viewed as a prejudge hint to explicitly elicit drivers to slow down to avoid failing

Key Step: The reasoning step where exists one next step that has no correct rationales
Prejudge Hint: The denominator cannot be 0

Note: When LLM finishes step 1, the searching procedure will be performed and find that there exists Step 2 that has no correct rationale, so it should provide prejudge hint after Step 1

Figure 1: Examples of prejudge in the scenarios of the real world and LLM reasoning.

1 Introduction

Large language models (LLMs) have made great inroads in solving natural language processing (NLP) tasks (Brown et al., 2020; Ouyang et al., 2022; OpenAI, 2023), but still struggle to produce accurate answers to complex reasoning problems. Prior research tackles this challenge by designing well-crafted prompts to elicit the LLM to follow a step-by-step thinking paradigm, such as in-context learning (Liu et al., 2023), chain-of-thought (Wei et al., 2022; Wang et al., 2023b, 2024c), and agentic learning (Park et al., 2023). However, these approaches akin to the System 1-style *fast-thinking* paradigm (Kahneman, 2011) inevitably bring errors to the inherent steps, making it hard to generate accurate and complete solutions in one breath.

Inspired by human recognition of System 2 (Kahneman, 2011), which is denoted as a *slow-thinking* paradigm emulates human reasoning through a slower and deeper thought process (Zelikman et al., 2024), most of the works have unveiled that extending the reasoning with verification, critiquing, and refinement components can significantly enhance the reasoning capability (Cobbe et al., 2021; Lightman et al., 2024; Snell et al., 2024; Qi et al., 2024; Shinn et al., 2023; Gou et al., 2024; Plaat et al., 2024; Madaan et al., 2023). One major benefit is that these components can offer precise feedback, which is a valuable signal that enables the LLM to adapt or roll back the current of thought opportunely (Kumar et al., 2024; Wang et al., 2024e; Chen and Li, 2024). In addition, a series of research (e.g., OpenAI's o1 (Jaech et al., 2024), DeepSeek R1 (Guo et al., 2025)) has explored that post-training in the supervised fine-tuning (SFT) or reinforcement learning (RL) stage can inject these capabilities into model parameters and achieve high

[1] Code and data is released at https://github.com/xxx.

grades by scaling test-time cost, which spurred the development of System 2-like reasoning (Brown et al., 2024; Snell et al., 2024; Shao et al., 2024; Rafailov et al., 2023). Despite this success, the generated responses expose that LLMs tend to frequently prefer trial and error, leading to redundant error and reflection information, which is not very advocated in human consciousness.

In this paper, we introduce a new thought mode named *process prejudge* [2], which is defined as prior consideration or judgment about what is about to happen in the subsequence reasoning steps. In the real world, this capability aims to help people learn from past experiences and improve the accuracy of each thinking step when solving similar problems in the future. It is usually acquired after repeated interaction with the environment. Take a vivid example illustrated in Figure 1, when driving a vehicle and reaching the entrance of a curve close to a cliff, an experienced driver will slow down in advance. This is a prejudge action based on the experience that the vehicle will fall off the cliff due to inertia. Therefore, a natural question arises: *is process prejudge useful for LLM in reasoning scenarios?*

To reach this goal, we first define a *prejudge node* in the rationale, which is a specific reasoning step, and at least one step follows the prejudge node that has no path toward the correct answer. For example in Figure 1, the LLM may make mistakes at "Step 2" and it can be prevented when prompted with a prejudge hint as "The denominator cannot be 0". To synthesize large-scale step-by-step reasoning data with *process prejudge*, we then propose an automatic reasoning framework with a dynamic tree-searching strategy, which is similar to Monto Carlo Tree Search (MCTS) (Kocsis and Szepesvári, 2006; Silver et al., 2016) but needs only one LLM to perform thinking, critiquing, prejudging and verifying during searching.

Ultimately, we construct 234k data from multiple open-source datasets to train the LLM with SFT and RL techniques. The extensive experiments conducted on mathematics and logic reasoning demonstrate that the paradigm of *prejudge before use* can substantially boost the LLM's reasoning ability.

2 Preliminary

2.1 LLM Reasoning with Textual Rationale

Given a LLM $\pi_\theta(\cdot)$ which is a transformer-based pre-trained model to map the input prompt to generated text, where θ is the parameters. For the reasoning task, given a question $\mathcal{Q}$, the LLM can provide a step-by-step reasoning chain consisting of T intermediate step, and the entire chain can be formed as $\mathcal{Z} = [z_1, \cdots, z_T]$, where z_i ($i \in [1, T]$) means the specific step[3]. The reasoning chain can be step-by-step generated by the LLM as $z_i = \pi_\theta(\mathcal{I} = \mathbb{I}(\mathcal{Q}, z_1, \cdots, z_{i-1}))$, where $\mathbb{I}(\cdots)$ is function to concate all generated prefix sequences to form the input prompt $\mathcal{I}$.

2.2 Bootstrapping with Tree Searching

Suppose that the prefix reasoning steps of $\mathcal{Q}$ are $\mathcal{Z}_{1:i-1} = [z_1, \cdots, z_{i-1}]$, the set of the next steps can be obtained by repeated sampling with the greedy method as $\{z_{ij} | z_{ij} \sim \pi_\theta(\mathcal{I} = \mathbb{I}(\mathcal{Q}, z_1, \cdots, z_{i-1}))\}$. Tree searching is an iterative generation process via repeatedly concatenating each sampled next step with a prefix sequence to form a new sequence before the next repeated sampling. Thus, the tree generated from z_{i-1} can be formed as $\mathcal{T}_{z_{i-1}}$. In this tree, z_{i-1} represents the root node in the first layer, and each node in the second layer is the child node of z_{i-1} denoted as $\{z_{ij}\}_{j=1}^N$, which is also the root node in the corresponding tree $\mathcal{T}_{z_{ij}}$, N is the number of repeated sampling at each layer. The searching process stops when the final answer is generated, and the last reasoning step can be viewed as the leaf node. Similar to MCTS, each node has a corresponding value score denoted as $v(\cdot)$ that represents the potential that reaches the correct answer.

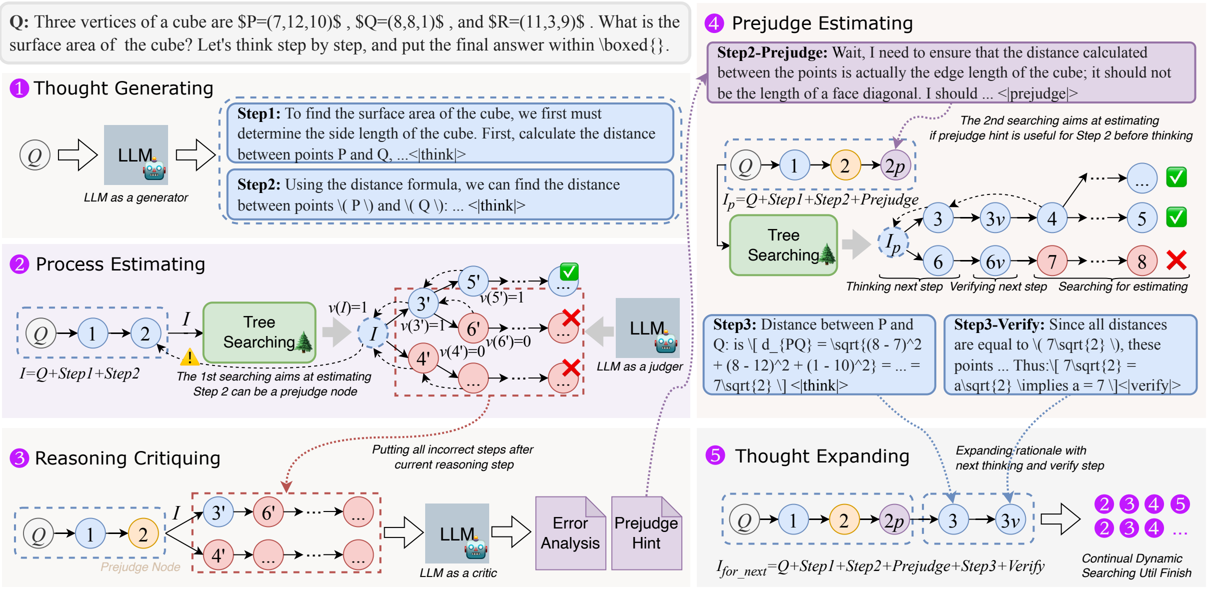
3 Methodology

In this section, we introduce a new reasoning method named *Prejudge Before Think* (PBT), which aims to elicit the LLM pauses to deeply consider what errors will occur and how to bypass them instead of excessive trial and error. Hence, we pose three questions to illustrate how our approach works:

- **R1**: Where should the LLM pause to make a prejudgement when reasoning?

[2] Notely, the "prejudge" in this paper means the "predictive ability" instead of "prejudice", we use the word "prejudge" aim to distinguish it from "predict" in machine learning.

[3] To elicit the LLM to generate this chain, the question $\mathcal{Q}$ should be articulated through a well-crafted instruction prompt or template. We omit this component to minimize the use of variables in writing.



Q: Three vertices of a cube are $P=(7,12,10)$ , $Q=(8,8,1)$ , and $R=(11,3,9)$ . What is the surface area of the cube? Let's think step by step, and put the final answer within \boxed{}.

Step1: To find the surface area of the cube, we first must determine the side length of the cube. First, calculate the distance between points P and Q, ...<|think|>

Step2: Using the distance formula, we can find the distance between points \(P \) and \(Q \): ... <|think|>

Step2-Prejudge: Wait, I need to ensure that the distance calculated between the points is actually the edge length of the cube; it should not be the length of a face diagonal. I should ... <|prejudge|>

Step3: Distance between P and Q: is \[d_{PQ} = \sqrt{(8 - 7)^2 + (8 - 12)^2 + (1 - 10)^2} = ... = 7\sqrt{2} \] <|think|>

Step3-Verify: Since all distances are equal to \(7\sqrt{2} \), these points ... Thus:\[7\sqrt{2} = a\sqrt{2} \implies a = 7 \]<|verify|>

Figure 2: The automated reasoning framework for synthesizing process prejudge with the dynamic tree-searching.

- **R2**: How to obtain the trajectory with PBT automatically without any external annotation?
- **R3**: How to boost the LLM reasoning capability with post-training techniques?

3.1 Prejudge Node in Rationale

To answer the first question, we observe that humans can use historical experience to help them make prejudgements before facing potential risks (Kahneman, 2011). Likewise, the time before errors occur is more suitable for stimulating the ability of LLMs to prejudge. Hence, we define this position as *prejudge node*.

Formally, given a question $\mathcal{Q}$ and the corresponding prefix reasoning steps $\mathcal{Z}_{1:i}$. To detect whether the LLM needs to make prejudgement after the step z_i, we can perform tree searching from this step to construct a tree denoted as $\mathcal{T}_{z_i}$, the value of each tree node can be obtained by hard estimation (Wang et al., 2024e). Specifically, we first use LLM-as-a-judger to check whether the final answer is correct, and the value of each leaf node can be set as 1 (or 0) if the result is correct (or incorrect). Then, the value of each internal node z_k can be backtracked from the leaf node as:

$$v(z_k) = \begin{cases} \text{Judger}(z_k), & \text{if } k = T \\ max(\{v(z_{k+1j})\}_{j=1}^{N})), & \text{if } k < T \end{cases} \tag{1}$$

where $\text{Judger}(\cdot) \in \{0, 1\}$ is the function of LLM-as-a-judger, z_{k+1j} is the child node of z_k.

Based on this value score, we can define a *prejudge value* that represents whether LLM should make prejudgement at the step z_i, which can be calculated as:

$$v_p(z_i) = v(z_i) \times \mathbf{1}(min(\{v(z_{i+1j})\}_{j=1}^{N}) = 0), \tag{2}$$

where $v_p(\cdot)$ is the prejudge value and the step z_i is a prejudge node when $v_p(z_i) = 1$, $\mathbf{1}(\cdot)$ is the indicator function, z_{i+1j} is the child node of z_i. Through this definition, at least one step follows the prejudge node and has no path toward the correct answer. This indicates that the LLM may make errors in the future so that needs to be prejudged.

3.2 Dynamic Tree Searching

We thus introduce how to synthesize the reasoning data with prejudge. We present a dynamic tree searching strategy in the reasoning framework, which enables only one LLM to find the prejudge position, generate critical information, and perform deep thought. The whole framework is shown in Figure 2, consisting of five stages: thought generating, process estimating, reasoning critiquing, prejudge estimating and thought expanding.

Thought Generating Given a question, we first to let the LLM generate a few steps without bootstrapping. Inspired by (Wang and Zhou, 2024), the start sequence has a greater impact on the performance of subsequent reasoning, especially for smaller models, so a lower temperature coefficient is selected to ensure the accuracy of reasoning in the first few steps. By default, we urge the model to generate two steps as the prefix sequence. Each thinking step will be ended with a special tag as "<|think|>".

Process Estimating Once the prefix sequence is generated, we improve the temperature coefficient and perform the first tree-searching process. This tree-searching aims to estimate whether the current step is a prejudge node, we can obtain the corresponding prejudge value v_p. Specifically, if the prejudge value is 0, it means that there is no need to make prejudgement here, we can randomly select one child node and concatenate it with the prefix sequence for the next iteration. Otherwise, we will continue the following process to obtain the prejudge hint before the next thinking step.

Reasoning Critiquing This stage aims to generate error analysis based on the incorrect reasoning path derived from the tree search. Specifically, we concatenate the question and prefix sequence with all incorrect reasoning paths to form a prompt and use the LLM as a critic (Shinn et al., 2023; Gou et al., 2024) to generate the corresponding error analysis. Generally, error analysis summarizes and induces existing reasoning results, making all reasoning steps visible. In contrast, LLM prejudging guesses possible errors without examining subsequent reasoning steps. To achieve this goal, we design an instruction prompt for the LLM to generate a prejudging hint based on the error analysis, and this information will be used to prompt the LLM to avoid errors. The prejudging hint can be tagged with a "<|prejudge|>" token at the end of the text. The specific prompt and generated examples are shown in Appendix C.3.

Prejudge Estimating After the prejudge hint is generated, we aim to evaluate its usefulness for the upcoming reasoning steps. Specifically, we perform tree-searching for the second time. Unlike the tree-searching conducted during the process estimation stage, the requirements of the generated response consist of three components: i) the next step thought (tagging with "<|think|>"): we expect the LLM to reconsider the next reasoning step in light of the prejudge hint; ii) the verification of the next step thought (tagging with "<|verify|>"): we introduce an explicit verification step for the LLM to check if the new thought aligns with the prejudge hint; and iii) the remaining steps with bootstrapping (tagging with "<|think|>"): we employ tree-searching to sample all reasoning steps to determine if the prejudge hint enables the LLM to arrive at the correct answer, utilizing the rejected sampling method to select the appropriate prejudge hint. We have crafted an instruction prompt to encourage the LLM to generate the aforementioned information, with the prompt and generated examples displayed in Appendix C.4.

Algorithm 1 Algorithm for Prejudge Reasoning

```
Require: LLM $\pi_\theta$, question $Q$, and prompt function $\mathbb{I}(\cdot)$.
Generating the first step via $z_0 = \pi_\theta(\mathbb{I}(Q))$;
for iteration $i$ in $[0, \cdots, T]$ do
    Expanding the next steps $\{z_{ij}|z_{ij} \sim \pi_\theta(\mathcal{I} = \mathbb{I}(\mathcal{Q}, z_1, \cdots, z_{i-1}))\}$;
    Process Estimating: Perform Judgement to detect whether each of next steps are prejudge node.
    Reasoning Critiquing: Generate self-critique information to guide the LLM to reach correct answer in the next step;
    Prejudge Estimating: Estimate whether the whole reasoning chains is correct.
end for
Obtain all training data and perform the first-time SFT, obtaining the model $\pi_{\theta_1}$.
Use $\pi_{\theta_1}$ to distill a large-scale prejudge-like data, and perform the second-time SFT and RL training, obtaining the model $\pi_{\theta_2}$.
return The LLM model $\pi_{\theta_2}$.
```

Thought Expanding Finally, we expand the reasoning chain with a prejudge hint, the selected next step consideration, and the corresponding verification. Then, we continue the next iteration of dynamic searching until we reach the final answer.

3.3 Two-phase Post-training

Dynamic tree searching involves significant test-time costs because it necessitates constructing multiple trees for each query. To equip the more efficient data synthesis pipeline with prejudge reasoning, we introduce a two-phase post-training strategy that allows for simultaneous data synthesis and model training. The whole algorithm is illustrated in Algorithm 1.

First Phase: Cold Start via Dynamic Searching In the first phase, the aim is to construct a small amount of data with prejudge reasoning. To collect this data, we choose a small instruct-based LLM to finish the dynamic tree-searching and obtain all correct rationales by rejected sampling. Specifically, we select multiple training sources from GSM8K (Cobbe et al., 2021), MATH (Hendrycks et al., 2021), SVAMP (Patel et al., 2021), AQuA (Ling et al., 2017), Numina Math (LI et al., 2024), American Invitational Mathematics Examination (AIME 1983∼2023) [4], PRM800K (Lightman et al., 2024), and MetaMath QA (Yu et al., 2024) and thus filter out about 21k complex queries for dynamic tree-searching, which can derive long CoT responses by zero-shot prompting (Kojima et al., 2022). Finally, we gather approximately 39k rationales and utilize these training samples to train a base LLM through SFT as the cold start model.

Second Phase: Distillation for Data Scaling The second phase focuses on self-evolution, aiming to leverage the SFT mode from the first phase to perform distillation. A large-scale, prejudge-style rationale will be constructed using a simple zero-shot prompting (Kojima et al., 2022). To enhance the quality of the rationale, we employ the self-consistency strategy to recall the most reliable rationale, which can be utilized as the prejudge estimation stage in dynamic tree-searching. We thus select the remaining queries from the training sources, obtaining approximately 195k rationales. During the training periods, we combine the curated data generated from both phases and retrain SFT on the base LLM. For the RL, we perform Group Relative Policy Optimization (GRPO) (Shao et al., 2024) and Directly Preference Optimization (DPO) (Rafailov et al., 2023) algorithms to investigate how performance improvement.

4 Experiments

4.1 Implementation Settings

In the first phase, we choose Qwen2.5-14B-Instruct (Team, 2024) as the small instruct-based LLM to perform dynamic tree-searching. By default, we use the "\n\n" as the step terminator, and the maximum sampling step length is 14 for each query. Complete implementation details of our dynamic tree-search algorithm are provided in Appendix B.1. For the cold start SFT, we choose Qwen2.5-7B/32B (Team, 2024) as the base LLM. In the second phase, we use the cold start 32B LLM to perform the distillation, and the number of repeated samples in Self-consistency is $N = 32$. For the SFT and RL, we choose Qwen2.5-32B as the base model.

4.2 Benchmarks and Evaluations

We select several competition-level reasoning benchmarks to demonstrate how LLM performance is improved with prejudge reasoning. These include GSM8K (Cobbe et al., 2021), MATH-500 (Lightman et al., 2024), AQuA (Ling et al., 2017), SVAMP (Patel et al., 2021), TheoremQA (Chen et al., 2023), AIME-2024 (MAA, 2024), GAOKAO-2023 (Zhang et al., 2023), and GPQA-Diamond (Rein et al., 2023). We follow previous works (Wang et al., 2024e) to use Qwen2.5-72B-Instruct (Team, 2024) as a judger to evaluate whether the generated answer matches the ground truth, and the metric is accuracy value.

4.3 Baselines

We select the following baselines to compare with our method: 1) Chain-of-Thought (CoT) Training, which aims to distill multiple rationales using CoT prompts for the supervised fine-tuning (SFT) training data. 2) Self-Refine (Kumar et al., 2024), which seeks to correct mistakes based on outcome- or process-based feedback. We obtain this rationale by prompting the small LLM with a well-designed zero-shot instruction. 3) PBT w/o. Verify, a variant version of our method that removes all verification components in dynamic tree searching; this means the LLM only makes prejudgments without any verification. 4) PBT w. CoT, which combines all data from prejudge and CoT. We also select GPT-4o (Hurst et al., 2024) and OpenAI's o1 (Jaech et al., 2024) as strong baselines to demonstrate state-of-the-art performance.

4.4 Main Results

Table 1 presents the performance comparison with multiple baselines on competition-level complex reasoning tasks. Through the results, we thus draw the following conclusion: 1) Our PBT consistently outperforms CoT training and Self-Refine across all benchmarks using both 7B and 32B backbones. Specifically, in the first phase, PBT achieves an average accuracy of 59.0% and 65.6% for Qwen2.5-7B and Qwen2.5-32B, representing improvements of 3.5% and 2.4% over CoT and 2.5% and 1.2% over Self-Refine. 2) The "verify" components in PBT are crucial for ensuring the accuracy of prejudgment. We can see that all results decline when this component is removed, except for Qwen2.5-32B on AIME-2024. 3) Combining CoT data with PBT can further enhance the accuracy of the 32B

[4] https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions.

Methods	**GSM8K**	**MATH 500**	**AQuA**	**SVAMP**	**Theorem QA**	**AIME 2024**	**GAOKAO 2023**	**GPQA Diamond**	**Avg.**
OpenAI's o1	-	94.8	-	-	-	74.4	-	77.3	-
GPT-4o	94.2	76.8	-	93.9	49.7	9.3	88.1	50.6	-
DeepSeek R1	-	97.3	-	-	-	79.8	-	71.5	-
dist. Qwen-7B	-	92.8	-	-	-	55.5	-	49.1	-
dist. Qwen-32B	-	94.3	-	-	-	72.6	-	62.1	-
Base Model: Qwen2.5-7B									
CoT$_{\#1}$	84.6	65.2	67.7	89.0	39.3	6.7	71.4	19.7	55.5
Self-Refine$_{\#1}$	85.1	66.4	68.3	90.7	40.6	6.7	71.0	23.5	56.5
PBT$_{\#1}$	**87.6**	**68.0**	**70.1**	90.3	**41.4**	**13.3**	**73.5**	**27.8**	**59.0**
w/o. Verify	85.9	67.4	68.9	89.7	41.1	10.0	73.1	25.6	57.7
w. CoT	85.6	67.2	68.5	**91.7**	41.0	13.3	73.4	26.2	58.4
PBT$_{\#2}$	**89.4**	**72.6**	**71.7**	**92.0**	**43.2**	**16.7**	**75.5**	**31.3**	**61.6**
Base Model: Qwen2.5-32B									
CoT$_{\#1}$	91.7	77.6	75.2	89.3	48.7	6.7	77.6	38.4	63.2
Self-Refine$_{\#1}$	92.6	77.3	76.8	90.4	49.3	10.0	79.4	39.4	64.4
PBT$_{\#1}$	92.4	**78.2**	**80.3**	91.0	50.6	13.3	78.6	**40.2**	65.6
w/o. Verify	92.1	77.8	79.1	90.7	50.0	13.3	78.3	39.5	65.1
w. CoT	**93.9**	77.8	79.2	**92.3**	**51.8**	**16.7**	**83.7**	38.9	**66.8**
PBT$_{\#2}$	**93.1**	**81.0**	**81.7**	**93.7**	**52.0**	**20.0**	**85.6**	**47.7**	**69.4**

Table 1: Main results (%) over multiple complex reasoning tasks. **PBT** (prejudge before think) is our method, and the subscript $_{\#1}$ and $_{\#2}$ denotes the first phase and second phase, respectively.

Methods	**PBT**$_{\#1}$	**w. DPO**	**w. GRPO**
GSM8K	87.6	90.5	91.7
MATH500	68.0	70.4	71.2
AQuA	70.1	74.0	74.8
SVAMP	90.3	91.5	92.2
Theorem QA	41.4	44.8	45.3
AIME2024	13.3	16.7	26.7
GAOKAO2023	73.5	78.4	79.3
GPQA-Diamond	27.8	28.6	30.1
Avg.	59.0	61.9	63.9

Table 2: The improvement (%) of RL for Qwen2.5-7B.

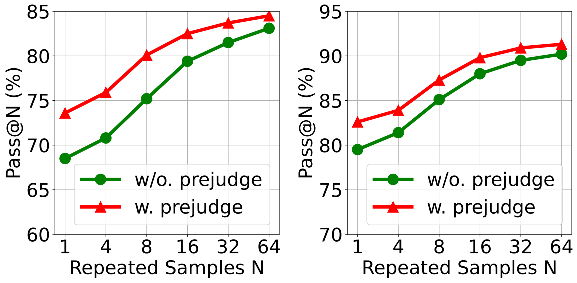


Figure 3: Effect of prejudge in complex reasoning with Qwen2.5-7B-Instruct (Left) and Qwen2.5-32B-Instruct (Right).

model. We find that **PBT**$_{\#1}$ w. CoT outperforms **PBT**$_{\#1}$ by 1.2%, indicating that larger models can benefit from diverse rationales, enabling them to utilize various styles of rationales to solve problems. 4) The two-phase strategy can bring obvious improvement. Compared with **PBT**$_{\#1}$, **PBT**$_{\#2}$ can further improve by 3%, demonstrating the effectiveness of two-phase post-training strategy.

5 Analysis

5.1 Performance Improvement of RL

To investigate performance improvements when applying RL with pre-judgment reasoning, we utilize two RL techniques to further enhance the SFT model in the first phase. Results shown in Table 2 demonstrate that the two RL methods substantially outperform the SFT. GRPO achieves the best performance, surpassing DPO by 2.0%, indicating that online RL is more effective in boosting reasoning ability than the offline method.

5.2 Effect of Prejudge in Test-time

To explore why prejudgment is effective for complex reasoning, we designed a thought completion task that demonstrates how the LLM reasons with prejudgment hints. Specifically, we randomly select 2,000 queries from the second phase that do not appear in the first phase and keep only the first prejudge node with the prejudge hint and the prefix-generated steps. For each query, we can obtain two incomplete responses: one that contains only the thinking step (i.e., remove the prejudge hint) and the other that contains the prejudge hint (i.e.,

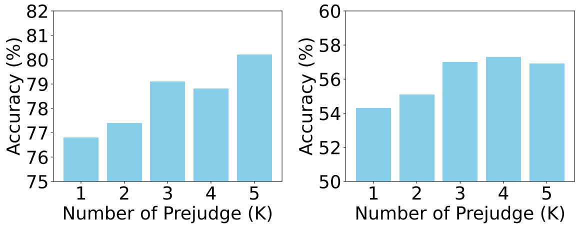


Figure 4: Effect of the number of prejudge hints over GSM8K (Left) and MATH500 (Right).

Methods	LIMO	LIMO + PBT	Gain
GSM8K	95.1	95.5	+0.4
MATH500	94.8	94.0	-0.8
AQuA	86.9	87.8	+0.9
SVAMP	91.3	92.3	+1.0
Theorem QA	54.6	58.0	+4.6
AIME2024	57.1	54.6	-2.5
GAOKAO2023	81.0	83.6	+2.6
GPQA-Diamond	66.7	63.0	-3.7

Table 3: The performance (%) with o1-like data.

Methods	PBT$_{\#1}$	MCTS	Math-Shepherd
GSM8K	**87.6**	79.50	85.1
MATH500	**68.0**	53.1	66.2
AIME2024	**13.3**	6.7	13.3
GPQA-Diamond	**27.8**	19.5	26.2

Table 4: The comparison of performance (%) with Math-Shepherd and MCTS.

maintain the prejudge hint). We choose Qwen2.5-7B/32B-Instruct as the LLM. To observe the performance, we let the LLM complete the reasoning steps with only the prompt "Let's think step by step" concatenated with the query and prefix-generated steps. We then draw a curve to see the performance when the test time scales up. Figure 3 shows that the Pass@N value is, on average, 3% higher with prejudge hints than without, suggesting that the rationale provided by the prejudge hint can better assist the LLM in avoiding mistakes.

5.3 Effect of the Number of Prejudge

In this section, we explore what's the effect of the number of prejudge in LLM reasoning. We sample five different sets $\mathcal{D}_k$ from the synthesized data in the first phase for SFT training, where $k \in \{1, 2, 3, 4, 5\}$ denotes the number of the prejudge node in the corresponding rationale. In other words, each rationale in $\mathcal{D}_k$ has only k prejudge hint. We perform data processing to ensure that the number of queries and rationales in each set is consistent. To this end, each set has about 9k examples for SFT training. The results displayed in Figure 4 suggest that adding the number of prejudge hints into the training data can significantly improve the reasoning ability of LLM. We believe that the increase in the number of prejudges will indirectly increase the overall length of rationale, which can further improve the certainty of LLM's output when thinking about problems (Jaech et al., 2024; Guo et al., 2025). In addition, the more prejudges there are, the more likely the model will make prejudges, which can better guide the model to make incorrect prejudges before thinking.

5.4 Compatibility with o1-like Reasoning

We end this section by investigating the performance of mixing with o1-like training data. We select data that has been widely used recently from LIMO (Ye et al., 2025), which utilizes less data to achieve optimal performance on competition-level tasks. We gather all queries from AIME (1983∼2023) to create prejudge reasoning data through dynamic tree searching and blend them with LIMO. As shown in Table 3, we can obtain the following suggestion: Although prejudge before use is not entirely o1-like data, simply mixing them can still maintain very high performance, and some benchmarks can be further improved. This shows that prejudge can be better integrated into o1-like reasoning, which also provides a new mode as a reference for o1-like reasoning community.

5.5 Effectiveness of Dynamic Tree-searching

We provide a comparison with some related searching methods, such as MCTS, Math-Shepherd. In the experiment, MCTS is directly used for the sampling generation of the model, while Math-Shepherd directly reproduces the reward model training and RL enhancement used in the original paper. Finally, it is compared with PBT. The result is shown in Table 4, our PBT can obtain the best performances. This suggests that moving the sampling phase forward to training can significantly improve the inference performance of the model, surpassing the performance of using tree sampling in the inference phase.

6 Related Works

LLM Reasoning by Learning From Mistakes Developing the LLM with capabilities for correcting, reflecting, critiquing, and verifying has been one of the essential strategies for enhancing the LLM's reasoning ability. The essence of these methods is to learn from mistakes. Previous works aim to design zero-shot prompts or few-shot examples to encourage the LLM to utilize external feedback (Madaan et al., 2023; Welleck et al., 2023; Xi et al., 2023; Wang et al., 2024b). However, these methods heavily rely on external feedback and limit the model's ability to think spontaneously. To remedy this dilemma, most recent works focus on post-training by injecting these abilities (i.e., correcting, reflecting, critiquing, and verifying) into model's parameters (Gao et al., 2024a; Wang et al., 2023a; Zhou et al., 2024). Another line of research leverages self-training ways to develop these capabilities (Qu et al., 2024; Kumar et al., 2024; Zheng et al., 2024; Xi et al., 2024). Unlike them, we focus on the ability to prejudge, which helps the LLM take a moment to consider potential mistakes and think about how to avoid them before acting. Prejudging is also a way to learn from mistakes without trial and error.

Post-training in LLM Reasoning With the development of OpenAI's o1 (Jaech et al., 2024) and Deepseek R1 (Guo et al., 2025), the post-training with test-time scaling has been powerful and versatile techniques in reasoning enhancement. These studies typically increase inference computation by extending the model's thinking chains with tree search (Hao et al., 2023; Zhang et al., 2024; Zelikman et al., 2024; Nori et al., 2024; Gao et al., 2024b), process-based optimization (Uesato et al., 2022; Wang et al., 2024e; Lightman et al., 2024; Wang et al., 2024a), and self-play (Huang et al., 2023; Chen et al., 2024; Wang et al., 2024d; Wu et al., 2024). To enhance the reasoning ability of small language models, some recent works perform distillation by DeepSeek R1 on Qwen2.5-7B and achieve satisfactory performance (Wen et al., 2025; Ye et al., 2025). Unlike them, we use a small Instruct-like LLM and propose a dynamic tree-searching algorithm to synthesize rationale with prejudgment and verification and develop a two-phase post-training strategy to enhance the model's reasoning ability. We also investigate the performance gains achieved by scaling up the testing time, which indicates that prejudging before thinking can effectively elicit the model to avoid mistakes.

7 Discussions

We thus provide discussions about this works.

The relation with Tree-of-Thoughts and MCTS For Tree-of-Thought (Yao et al., 2023) and MCTS (Kocsis and Szepesvári, 2006; Silver et al., 2016), which belong to System 1 mode, aim to improve the self-consistency method by ensembling multiple step-wise trajectories derived from tree-searching. Yet, our works focus on exploring a novel reasoning behavior in System 2 mode (i.e., prejudge reasoning) along with self-verify and reflection. Hence, we have very different research motivations. From the perspective of technique, we found that the Tree-of-thought and MCTS can only help us find the prejudge node through the estimated value, but the tree is still static so it cannot combine self-verify and reflection to dynamically adjust the reasoning path when meeting the prejudge node and obtain the corresponding prejudge hint. Therefore, we introduce a dynamic tree-searching algorithm, which can be viewed as the major extension in Tree-of-Thought.

The difference from PBT to Math-Shepherd For Math-Shepherd (Wang et al., 2024e) and our work, both of them use tree-searching method to estimate the value of each step, and we refer to the value estimation strategy in Math-Shepherd to help find the prejudge node. However, the main differences are still obvious: 1) Math-Shepherd focuses on using tree-search to form step-wise labels to train a PRM model, which can be viewed as a reward model or value model in RL, and the experiments demonstrate that the PRM model is effect for RL. 2) our PBT focuses on LLM (policy) itself by using the dynamic tree-searching algorithm to construct prejudge-like trajectories along with self-verify and reflection, and the two-stage training strategy is designed to elicit the prejudge behavior in LLM reasoning via SFT and RL to enhance the accuracy of complex tasks.

8 Conclusion

In this paper, we present a novel paradigm of "prejudge before thinking" inspired by System 2's slow thinking mode. We propose synthesizing training data using a dynamic tree-searching method with a small LLM and introduce a two-phase post-training strategy to enhance the model's reasoning ability with SFT and RL techniques. We conduct extensive experiments on multiple competition-level complex reasoning benchmarks, and the results demonstrate that the rationale embedded with the prejudged hints can guide the LLM to avoid making mistakes.

Limitations

Our paper has some limitations, which we leave for future work:

The computational cost of dynamic tree-searching The dynamic tree-searching algorithm incurs a high computational cost in the experiments. Each query takes about 5 minutes to generate four entire rationales. We attempted to use few-shot examples to prompt the strong LLM to generate the rationale with a preconceived notion, but we found that the preconceived notion was incorrect and the responses were not coherent. In the future, we will focus on the time efficiency on the searching strategy.

The reasoning format We found that recent research from Deepseek R1 suggests that large-scale reinforcement learning based on a backbone can stimulate the model's self-reflection and slow thinking style. In contrast, our work focuses on data synthesis to construct System 2-like data. However, this leads us to a research topic concerning how to selectively activate specific reasoning modes through the RL stage. In other words, can we design a reward function or other strategies that allow LLMs to make prejudgments and combine them with some novel modes (e.g., aha moments) to enhance reasoning capabilities?

Social Impact and Ethics

In terms of social impact, the reasoning data we utilize are all from publicly available data sources. Infusing this information into the model's reasoning process will not introduce additional bias. However, the open-source backbones we used may have some negative impacts, such as gender and social bias. Our work would unavoidably suffer from these issues. We suggest that users should carefully address potential risks when the proposed method is deployed online.

References

Bradley C. A. Brown, Jordan Juravsky, Ryan Saul Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. 2024. Large language monkeys: Scaling inference compute with repeated sampling. *CoRR*, abs/2407.21787.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *NeurIPS*.

Sijia Chen and Baochun Li. 2024. Toward adaptive reasoning in large language models with thought rollback. In *ICML*. OpenReview.net.

Wenhu Chen, Ming Yin, Max Ku, Pan Lu, Yixin Wan, Xueguang Ma, Jianyu Xu, Xinyi Wang, and Tony Xia. 2023. Theoremqa: A theorem-driven question answering dataset. In *EMNLP*, pages 7889–7901. Association for Computational Linguistics.

Zixiang Chen, Yihe Deng, Huizhuo Yuan, Kaixuan Ji, and Quanquan Gu. 2024. Self-play fine-tuning converts weak language models to strong language models. In *ICML*. OpenReview.net.

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.

Bofei Gao, Zefan Cai, Runxin Xu, Peiyi Wang, Ce Zheng, Runji Lin, Keming Lu, Junyang Lin, Chang Zhou, Wen Xiao, Junjie Hu, Tianyu Liu, and Baobao Chang. 2024a. LLM critics help catch bugs in mathematics: Towards a better mathematical verifier with natural language feedback. *CoRR*, abs/2406.14024.

Zitian Gao, Boye Niu, Xuzheng He, Haotian Xu, Hongzhang Liu, Aiwei Liu, Xuming Hu, and Lijie Wen. 2024b. Interpretable contrastive monte carlo tree search reasoning. *CoRR*, abs/2410.01707.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. 2024. CRITIC: large language models can self-correct with tool-interactive critiquing. In *ICLR*. OpenReview.net.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

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. In *EMNLP*, pages 8154–8173. 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. In *NeurIPS*.

Jiaxin Huang, Shixiang Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. 2023. Large language models can self-improve. In *EMNLP*, pages 1051–1068. Association for Computational Linguistics.

Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, Aleksander Madry, Alex Baker-Whitcomb, Alex Beutel, Alex Borzunov, Alex Carney, Alex Chow, Alex Kirillov, Alex Nichol, Alex Paino, Alex Renzin, Alex Tachard Passos, Alexander Kirillov, Alexi Christakis, Alexis Conneau, Ali Kamali, Allan Jabri, Allison Moyer, Allison Tam, Amadou Crookes, Amin Tootoonchian, Ananya Kumar, Andrea Vallone, Andrej Karpathy, Andrew Braunstein, Andrew Cann, Andrew Codispoti, Andrew Galu, Andrew Kondrich, Andrew Tulloch, Andrey Mishchenko, Angela Baek, Angela Jiang, Antoine Pelisse, Antonia Woodford, Anuj Gosalia, Arka Dhar, Ashley Pantuliano, Avi Nayak, Avital Oliver, Barret Zoph, Behrooz Ghorbani, Ben Leimberger, Ben Rossen, Ben Sokolowsky, Ben Wang, Benjamin Zweig, Beth Hoover, Blake Samic, Bob McGrew, Bobby Spero, Bogo Giertler, Bowen Cheng, Brad Lightcap, Brandon Walkin, Brendan Quinn, Brian Guarraci, Brian Hsu, Bright Kellogg, Brydon Eastman, Camillo Lugaresi, Carroll L. Wainwright, Cary Bassin, Cary Hudson, Casey Chu, Chad Nelson, Chak Li, Chan Jun Shern, Channing Conger, Charlotte Barette, Chelsea Voss, Chen Ding, Cheng Lu, Chong Zhang, Chris Beaumont, Chris Hallacy, Chris Koch, Christian Gibson, Christina Kim, Christine Choi, Christine McLeavey, Christopher Hesse, Claudia Fischer, Clemens Winter, Coley Czarnecki, Colin Jarvis, Colin Wei, Constantin Koumouzelis, and Dane Sherburn. 2024. Gpt-4o system card. *CoRR*, abs/2410.21276.

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, and Ilge Akkaya. 2024. Openai o1 system card. *CoRR*, abs/2412.16720.

Daniel Kahneman. 2011. Thinking, fast and slow. *Farrar, Straus and Giroux*.

Levente Kocsis and Csaba Szepesvári. 2006. Bandit based monte-carlo planning. In *ECML*, volume 4212 of *Lecture Notes in Computer Science*, pages 282–293. Springer.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. In *NeurIPS*.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D. Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, Lei M. Zhang, Kay McKinney, Disha Shrivastava, Cosmin Paduraru, George Tucker, Doina Precup, Feryal M. P. Behbahani, and Aleksandra Faust. 2024. Training language models to self-correct via reinforcement learning. *CoRR*, abs/2409.12917.

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.

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 *ICLR*. OpenReview.net.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. 2017. Program induction by rationale generation: Learning to solve and explain algebraic word problems. In *ACL*, pages 158–167. Association for Computational Linguistics.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pretrain, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Comput. Surv.*, 55(9):195:1–195:35.

MAA. 2024. American invitational mathematics examination - aime. *URL https://maa.org/math-competitions/american-invitational-mathematics-examination-aime*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. In *NeurIPS*.

Harsha Nori, Naoto Usuyama, Nicholas King, Scott Mayer McKinney, Xavier Fernandes, Sheng Zhang, and Eric Horvitz. 2024. From medprompt to o1: Exploration of run-time strategies for medical challenge problems and beyond. *CoRR*, abs/2411.03590.

OpenAI. 2023. GPT-4 technical report. *CoRR*, abs/2303.08774.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *NeurIPS*.

Joon Sung Park, Joseph C. O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *UIST*, pages 2:1–2:22. ACM.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are NLP models really able to solve simple math word problems? In *NAACL*, Online. Association for Computational Linguistics.

Aske Plaat, Annie Wong, Suzan Verberne, Joost Broekens, Niki van Stein, and Thomas Bäck. 2024. Reasoning with large language models, a survey. *CoRR*, abs/2407.11511.

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

Yuxiao Qu, Tianjun Zhang, Naman Garg, and Aviral Kumar. 2024. Recursive introspection: Teaching language model agents how to self-improve. In *NeurIPS*.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. *CoRR*, abs/2305.18290.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2023. GPQA: A graduate-level google-proof q&a benchmark. *CoRR*, abs/2311.12022.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *CoRR*, abs/2402.03300.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *NeurIPS*.

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.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. *CoRR*, abs/2408.03314.

Qwen Team. 2024. Qwen2. 5: A party of foundation models. *Qwen (Sept. 2024). url: https://qwenlm. github. io/blog/qwen2*, 5.

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

Chaojie Wang, Yanchen Deng, Zhiyi Lv, Zeng Liang, Jujie He, Shuicheng Yan, and Bo An. 2024a. Q*: Improving multi-step reasoning for llms with deliberative planning. *CoRR*, abs/2406.14283.

Jianing Wang, Qiushi Sun, Xiang Li, and Ming Gao. 2024b. Boosting language models reasoning with chain-of-knowledge prompting. In *ACL*, pages 4958–4981. Association for Computational Linguistics.

Jianing Wang, Junda Wu, Yupeng Hou, Yao Liu, Ming Gao, and Julian J. McAuley. 2024c. Instructgraph: Boosting large language models via graph-centric instruction tuning and preference alignment. In *ACL*, pages 13492–13510. Association for Computational Linguistics.

Jianing Wang, Yang Zhou, Xiaocheng Zhang, Mengjiao Bao, and Peng Yan. 2024d. Self-evolutionary large language models through uncertainty-enhanced preference optimization. *CoRR*, abs/2409.11212.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. 2024e. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. In *ACL*, pages 9426–9439. Association for Computational Linguistics.

Tianlu Wang, Ping Yu, Xiaoqing Ellen Tan, Sean O'Brien, Ramakanth Pasunuru, Jane Dwivedi-Yu, Olga Golovneva, Luke Zettlemoyer, Maryam Fazel-Zarandi, and Asli Celikyilmaz. 2023a. Shepherd: A critic for language model generation. *CoRR*, abs/2308.04592.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023b. Self-consistency improves chain of thought reasoning in language models. In *ICLR*. OpenReview.net.

Xuezhi Wang and Denny Zhou. 2024. Chain-of-thought reasoning without prompting. In *NeurIPS*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*.

Sean Welleck, Ximing Lu, Peter West, Faeze Brahman, Tianxiao Shen, Daniel Khashabi, and Yejin Choi. 2023. Generating sequences by learning to self-correct. In *ICLR*. OpenReview.net.

Liang Wen, Yunke Cai, Fenrui Xiao, Xin He, Qi An, Zhenyu Duan, Yimin Du, Junchen Liu, Lifu Tang, Xiaowei Lv, Haosheng Zou, Yongchao Deng, Shousheng Jia, and Xiangzheng Zhang. 2025. Light-r1: Curriculum sft, dpo and rl for long cot from scratch and beyond. *Preprint*, arXiv:2503.10460.

Yue Wu, Zhiqing Sun, Huizhuo Yuan, Kaixuan Ji, Yiming Yang, and Quanquan Gu. 2024. Self-play preference optimization for language model alignment. *CoRR*, abs/2405.00675.

Zhiheng Xi, Senjie Jin, Yuhao Zhou, Rui Zheng, Songyang Gao, Jia Liu, Tao Gui, Qi Zhang, and Xuanjing Huang. 2023. Self-polish: Enhance reasoning in large language models via problem refinement. In *EMNLP*, pages 11383–11406. Association for Computational Linguistics.

Zhiheng Xi, Dingwen Yang, Jixuan Huang, Jiafu Tang, Guanyu Li, Yiwen Ding, Wei He, Boyang Hong, Shihan Dou, Wenyu Zhan, Xiao Wang, Rui Zheng, Tao Ji, Xiaowei Shi, Yitao Zhai, Rongxiang Weng, Jingang Wang, Xunliang Cai, Tao Gui, Zuxuan Wu, Qi Zhang, Xipeng Qiu, Xuanjing Huang, and Yu-Gang Jiang. 2024. Enhancing LLM reasoning via critique models with test-time and training-time supervision. *CoRR*, abs/2411.16579.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *CoRR*, abs/2305.10601.

Yixin Ye, Zhen Huang, Yang Xiao, Ethan Chern, Shijie Xia, and Pengfei Liu. 2025. Limo: Less is more for reasoning. In *arXiv*. arXiv.

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 *ICLR*. OpenReview.net.

Eric Zelikman, Georges Harik, Yijia Shao, Varuna Jayasiri, Nick Haber, and Noah D. Goodman. 2024. Quiet-star: Language models can teach themselves to think before speaking. *CoRR*, abs/2403.09629.

Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. 2024. Rest-mcts*: LLM self-training via process reward guided tree search. In *NeurIPS*.

Xiaotian Zhang, Chunyang Li, Yi Zong, Zhengyu Ying, Liang He, and Xipeng Qiu. 2023. Evaluating the performance of large language models on GAOKAO benchmark. *CoRR*, abs/2305.12474.

Xin Zheng, Jie Lou, Boxi Cao, Xueru Wen, Yuqiu Ji, Hongyu Lin, Yaojie Lu, Xianpei Han, Debing Zhang, and Le Sun. 2024. Critic-cot: Boosting the reasoning abilities of large language model via chain-of-thoughts critic. *CoRR*, abs/2408.16326.

Aojun Zhou, Ke Wang, Zimu Lu, Weikang Shi, Sichun Luo, Zipeng Qin, Shaoqing Lu, Anya Jia, Linqi Song, Mingjie Zhan, and Hongsheng Li. 2024. Solving challenging math word problems using GPT-4 code interpreter with code-based self-verification. In *ICLR*. OpenReview.net.

Task	Domain	Source	Sampling	The 1st Phase		The 2nd Phase	
				#Search	#Train	#Search	#Train
GSM8K	MATH	(Cobbe et al., 2021)	7,473	7,473	7,473	0	0
MATH	MATH	(Hendrycks et al., 2021)	3,994	3,994	2,000	1,994	1,200
SVAMP	MATH	(Patel et al., 2021)	700	700	700	0	0
AQuA	MATH	(Ling et al., 2017)	97467	7,000	5,350	1,650	210
Numina Math	MATH	(LI et al., 2024)	34,473	12,000	4,800	22,473	15,800
AIME (1983∼2023)	MATH	(LI et al., 2024)	919	919	678	241	89
PRM800K	MATH	(Lightman et al., 2024)	12,000	12,000	7,800	4,200	1,390
MetaMath QA	MATH	(Yu et al., 2024)	150,000	0	0	150,000	63,077

Table 5: The data statistics of each task. The data for #Train is smaller than #Search because rejected sampling.

A Data Sources

For data collection, we select multiple training sources from GSM8K (Cobbe et al., 2021), MATH (Hendrycks et al., 2021), SVAMP (Patel et al., 2021), AQuA (Ling et al., 2017), Numina Math (LI et al., 2024), American Invitational Mathematics Examination (AIME 1983∼2023) [5], PRM800K (Lightman et al., 2024), and MetaMath QA (Yu et al., 2024). The details of each source is shown in Table 5.

B Experimental Setup Details

B.1 Details of Dynamic Tree-Searching

We develop a dynamic tree-searching to release process estimation and prejudge estimation. We first numbered and named the internal nodes of each tree. In order to facilitate tracing each node, we adopted a continuous coding strategy. For example, the node “2-4-1-3” is located at the fourth layer in the tree, and it is one of the child nodes of “2-4-1”. the reasoning step at the node “2-4-1-3” can be viewed as the third repeated sample generated from “2-4-1”.

Since tree search is an algorithm with exponentially increasing complexity, we agree that the number of repeated samplings at each layer is different and, finally, ensure that the number of paths (from root to all leaf nodes) does not exceed 1024.

C Prompt Engineering

C.1 Prompt Format

In this paper, we choose Qwen2.5-7B and Qwen2.5-32B as the backbone for post-training, so we design a new prompt format for the subsequence training. The format is:

```
<|im_start|>user
{Question}
Let's think step by step, and put
the final answer within \boxed{}.
<|im_end|>
<|im_start|>assistant
```

where “{Question}” is the placeholder for complex query, “<|im_start|>”, “<|im_end|>” are the special tokens in vocabulary set of Qwen2.5 model.

Prompt for LLM-as-a-Judger

Given a problem and the corresponding ground truths, the task is to verify if the generated answer can match one of the candidate ground truths. Please output "TRUE" or "FALSE" only.

Below is the one you need to verify:
### Start of Problem
{PROBLEM}
### End of Problem
### Start of Generated Answer
{FINAL_ANSWER}
### End of Generated Answer
### Start of Ground Truth
{GROUND_TRUTH}
### End of Ground Truth
### Start of Verification

Figure 5: The prompt for LLM-as-a-Judger.

C.2 Prompt for LLM-as-a-Judger

The prompt for making the LLM-as-a-Judger is shown in Figure 5. The prompt will be used in dynamic tree searching to detect whether each reasoning path is correct (i.e., matching the final answer in the box with the ground truth).

C.3 Prompt for LLM-as-a-Critic

The prompt for generating analysis and prejudge hint is shown in Figure 7. This prompt will be

[5] https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions.

Prompt for Prejudge Estimating

```
Question: {Question}
Let's use the self conversation to think step by step. Do not output '\\n\\n' at will, output '\\n\\n' only when each complete reasoning step is completed. After reasoning, please output the final answer in \\boxed.
If you see a prejudge prompt, you must first proceed the thinking step ONLY (please be careful to avoid the mistakes mentioned in the prejudge). Then verify on this new thinking step ONLY whether it is correct and successfully avoided the errors mentioned in the prejudge. If you find that there are still some errors, please rethink and improve it until you think it is correct. Lastly, continually completing the rest thinking steps. You must also maintain the conversation style like the previous thinking step.
The output format must be following:
### Thinking Only Next Step
...
### Verifying And Correcting Only Next Step
...
### Thinking The Rest Steps
...
```

Figure 6: The prompt for prejudge estimating.

utilized in dynamic tree searching to produce error analysis for all incorrect rationales and construct a prejudge hint for the prejudge node.

C.4 Prompt for Prejudge Estimating

The prompt for prejudge estimating is shown in Figure 6. When the prejudge hint is generated, we can use this prompt to elicit the LLM to generate the next thinking step, verify that step, and proceed with the remaining steps to reach the final answer.

D Case Study

To show how the LLM reasoning by prejudge, we provide two cases to demonstrate the rationale. The cases are shown in Figure 8 to Figure 10.

Prompt for LLM-as-a-Critic

You possess expertise in solving mathematical problems through a systematic, step-by-step reasoning process during which you are dedicated to preventing repeating any errors analyzed in experiences. Here is a problem and the corresponding correct answer:

Problem:
```
{Problem}
```
Correct Answers:
```
{Correct_Answer}
```

Now, I will give you the initialized reasoning solution steps, and some corresponding incorrect completions which aim at continually finishing the rest of the solution steps but reach the wrong answer. The reasoning situations are in the following:

Initialized Reasoning Steps:
```
{Prefix_Response}
```

{Suffix_Incorrect_Responses}

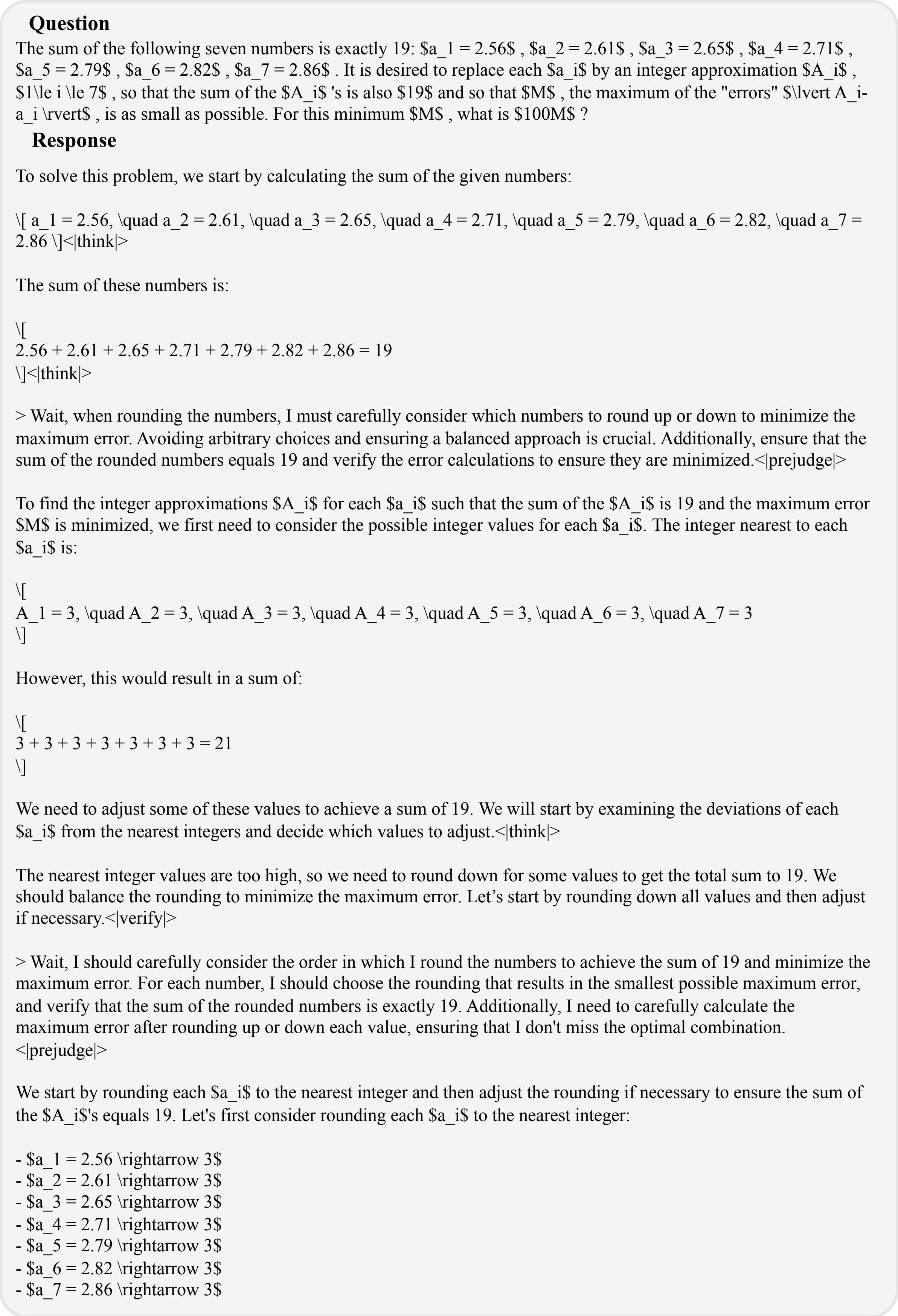
Please help me and give some following tips:

1) Errors Analysis: Each incorrect completion reaches an incorrect answer due to misconception, please list the specific mistakes details.
Cautions:
- DO NOT disclose the complete number (e.g., "Completion #1").

2) Prejudge: You will start reasoning from the given initialized step, please generate some detailed prejudge information to ask yourself to avoid making errors.
Cautions:
- The generated prediction information is intended to guide the next step of reasoning to avoid errors, it should be closed to the possible errors and the detailed error analysis;
- The generated prediction information prefers to tell yourself what mistakes to avoid, rather than remind yourself to verify, so DO NOT output any contents like "I should double-check ..." or "I need to verify ...";
- The generated prejudge should use coherent sentences without explicitly using line breaks or bold formatting for listing;
- When generating prejudge, please use a self-talk style with only one of the modal particles ("Wait", "Oh", "Hmmm", "Hold on") to perform smooth out.

Please note that the final output format must be in the following template:
### Errors Analysis
...
### Prejudge
...

Figure 7: The prompt for LLM-as-a-Critic.



Question

The sum of the following seven numbers is exactly 19: $a_1 = 2.56$ , $a_2 = 2.61$ , $a_3 = 2.65$ , $a_4 = 2.71$ , $a_5 = 2.79$ , $a_6 = 2.82$ , $a_7 = 2.86$ . It is desired to replace each $a_i$ by an integer approximation $A_i$ , $1\le i \le 7$ , so that the sum of the $A_i$ 's is also $19$ and so that $M$ , the maximum of the "errors" $\lvert A_i-a_i \rvert$ , is as small as possible. For this minimum $M$ , what is $100M$ ?

Response

To solve this problem, we start by calculating the sum of the given numbers:

\[a_1 = 2.56, \quad a_2 = 2.61, \quad a_3 = 2.65, \quad a_4 = 2.71, \quad a_5 = 2.79, \quad a_6 = 2.82, \quad a_7 = 2.86 \]<|think|>

The sum of these numbers is:

\[
2.56 + 2.61 + 2.65 + 2.71 + 2.79 + 2.82 + 2.86 = 19
\]<|think|>

> Wait, when rounding the numbers, I must carefully consider which numbers to round up or down to minimize the maximum error. Avoiding arbitrary choices and ensuring a balanced approach is crucial. Additionally, ensure that the sum of the rounded numbers equals 19 and verify the error calculations to ensure they are minimized.<|prejudge|>

To find the integer approximations $A_i$ for each $a_i$ such that the sum of the $A_i$ is 19 and the maximum error $M$ is minimized, we first need to consider the possible integer values for each $a_i$. The integer nearest to each $a_i$ is:

\[
A_1 = 3, \quad A_2 = 3, \quad A_3 = 3, \quad A_4 = 3, \quad A_5 = 3, \quad A_6 = 3, \quad A_7 = 3
\]

However, this would result in a sum of:

\[
3 + 3 + 3 + 3 + 3 + 3 + 3 = 21
\]

We need to adjust some of these values to achieve a sum of 19. We will start by examining the deviations of each $a_i$ from the nearest integers and decide which values to adjust.<|think|>

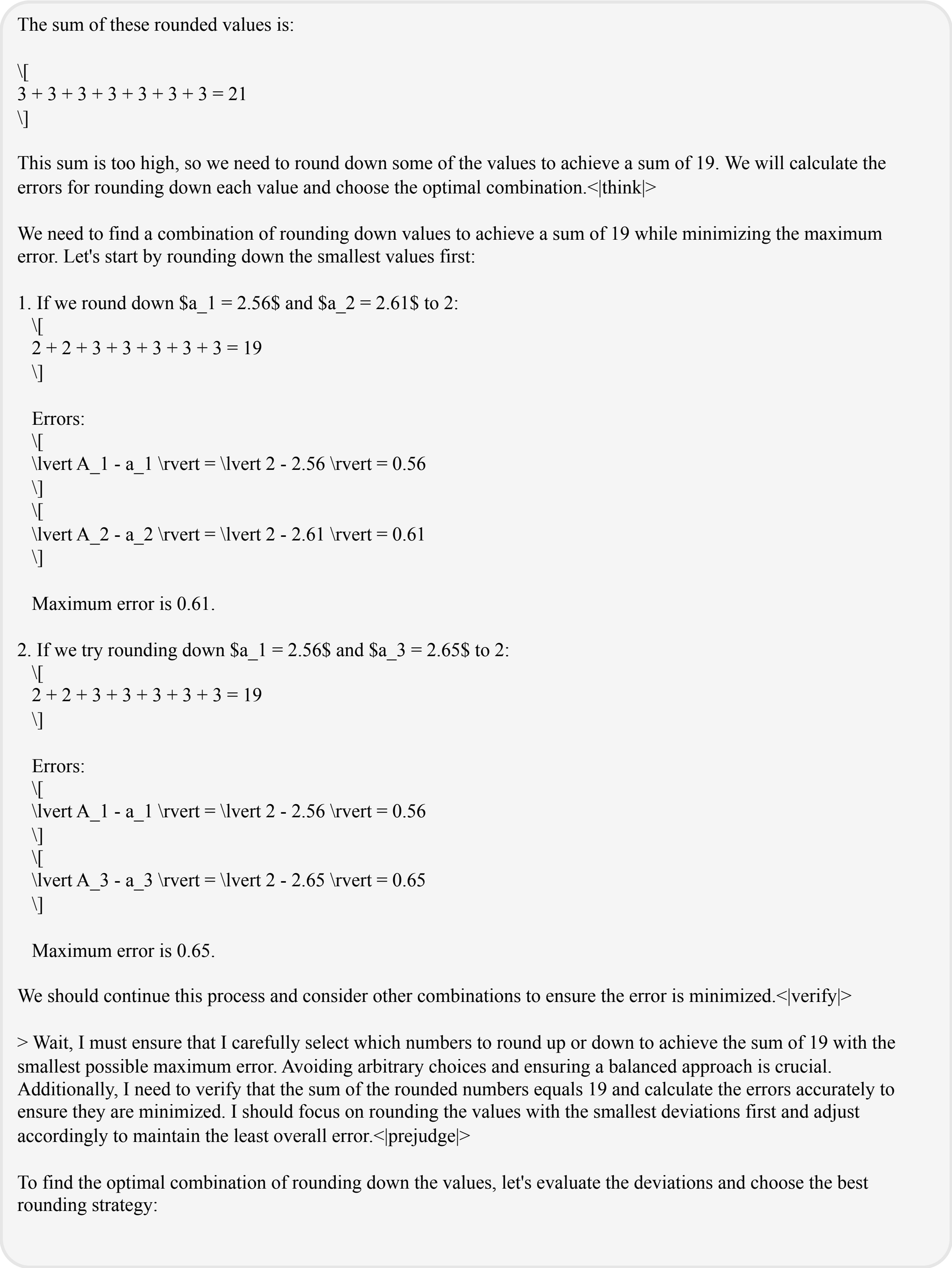
The nearest integer values are too high, so we need to round down for some values to get the total sum to 19. We should balance the rounding to minimize the maximum error. Let's start by rounding down all values and then adjust if necessary.<|verify|>

> Wait, I should carefully consider the order in which I round the numbers to achieve the sum of 19 and minimize the maximum error. For each number, I should choose the rounding that results in the smallest possible maximum error, and verify that the sum of the rounded numbers is exactly 19. Additionally, I need to carefully calculate the maximum error after rounding up or down each value, ensuring that I don't miss the optimal combination.
> <|prejudge|>

We start by rounding each $a_i$ to the nearest integer and then adjust the rounding if necessary to ensure the sum of the $A_i$'s equals 19. Let's first consider rounding each $a_i$ to the nearest integer:

- $a_1 = 2.56 \rightarrow 3$
- $a_2 = 2.61 \rightarrow 3$
- $a_3 = 2.65 \rightarrow 3$
- $a_4 = 2.71 \rightarrow 3$
- $a_5 = 2.79 \rightarrow 3$
- $a_6 = 2.82 \rightarrow 3$
- $a_7 = 2.86 \rightarrow 3$

Figure 8: The case of prejudge reasoning (part 1).



The sum of these rounded values is:

\[
3 + 3 + 3 + 3 + 3 + 3 + 3 = 21
\]

This sum is too high, so we need to round down some of the values to achieve a sum of 19. We will calculate the errors for rounding down each value and choose the optimal combination.<|think|>

We need to find a combination of rounding down values to achieve a sum of 19 while minimizing the maximum error. Let's start by rounding down the smallest values first:

1. If we round down $a_1 = 2.56$ and $a_2 = 2.61$ to 2:
\[
2 + 2 + 3 + 3 + 3 + 3 + 3 = 19
\]

Errors:
\[
\lvert A_1 - a_1 \rvert = \lvert 2 - 2.56 \rvert = 0.56
\]
\[
\lvert A_2 - a_2 \rvert = \lvert 2 - 2.61 \rvert = 0.61
\]

Maximum error is 0.61.

2. If we try rounding down $a_1 = 2.56$ and $a_3 = 2.65$ to 2:
\[
2 + 2 + 3 + 3 + 3 + 3 + 3 = 19
\]

Errors:
\[
\lvert A_1 - a_1 \rvert = \lvert 2 - 2.56 \rvert = 0.56
\]
\[
\lvert A_3 - a_3 \rvert = \lvert 2 - 2.65 \rvert = 0.65
\]

Maximum error is 0.65.

We should continue this process and consider other combinations to ensure the error is minimized.<|verify|>

> Wait, I must ensure that I carefully select which numbers to round up or down to achieve the sum of 19 with the smallest possible maximum error. Avoiding arbitrary choices and ensuring a balanced approach is crucial. Additionally, I need to verify that the sum of the rounded numbers equals 19 and calculate the errors accurately to ensure they are minimized. I should focus on rounding the values with the smallest deviations first and adjust accordingly to maintain the least overall error.<|prejudge|>

To find the optimal combination of rounding down the values, let's evaluate the deviations and choose the best rounding strategy:

Figure 9: The case of prejudge reasoning (part 2).

```
1. If we round down $a_1 = 2.56$ and $a_2 = 2.61$ to 2:
   \[
   \lvert A_1 - a_1 \rvert = \lvert 2 - 2.56 \rvert = 0.56
   \]
   \[
   \lvert A_2 - a_2 \rvert = \lvert 2 - 2.61 \rvert = 0.61
   \]
   Maximum error is 0.61.

2. If we round down $a_1 = 2.56$ and $a_3 = 2.65$ to 2:
   \[
   \lvert A_1 - a_1 \rvert = \lvert 2 - 2.56 \rvert = 0.56
   \]
   \[
   \lvert A_3 - a_3 \rvert = \lvert 2 - 2.65 \rvert = 0.65
   \]
   Maximum error is 0.65.

3. If we round down $a_2 = 2.61$ and $a_3 = 2.65$ to 2:
   \[
   \lvert A_2 - a_2 \rvert = \lvert 2 - 2.61 \rvert = 0.61
   \]
   \[
   \lvert A_3 - a_3 \rvert = \lvert 2 - 2.65 \rvert = 0.65
   \]
   Maximum error is 0.65.<|think|>

We see that rounding down $a_1 = 2.56$ and $a_2 = 2.61$ to 2 results in a maximum error of 0.61, which is the
smallest among the evaluated combinations. Let's verify if there are any other better combinations:

- Rounding down $a_1 = 2.56$ and $a_4 = 2.71$ to 2:
  \[
  \lvert A_1 - a_1 \rvert = \lvert 2 - 2.56 \rvert = 0.56
  \]
  \[
  \lvert A_4 - a_4 \rvert = \lvert 2 - 2.71 \rvert = 0.71
  \]
  Maximum error is 0.71.

- Rounding down $a_2 = 2.61$ and $a_4 = 2.71$ to 2:
  \[
  \lvert A_2 - a_2 \rvert = \lvert 2 - 2.61 \rvert = 0.61
  \]
  \[
  \lvert A_4 - a_4 \rvert = \lvert 2 - 2.71 \rvert = 0.71
  \]
  Maximum error is 0.71.

The combination of rounding down $a_1 = 2.56$ and $a_2 = 2.61$ to 2 yields the smallest maximum error of 0.61.
<|verify|>

Therefore, the smallest possible maximum error \( M \) is \( 0.61 \).

To find \( 100M \):<|think|>

\[
100M = 100 \times 0.61 = 61
\]

Thus, the final answer is \(\boxed{61}\).<|think|>
```

Figure 10: The case of prejudge reasoning (part 3).