

# CAN 1B LLM SURPASS 405B LLM? RETHINKING COMPUTE-OPTIMAL TEST-TIME SCALING

Anonymous authors

Paper under double-blind review

## ABSTRACT

Test-Time Scaling (TTS) is an important method for improving the performance of Large Language Models (LLMs) by using additional computation during the inference phase. However, current studies do not systematically analyze how policy models, Process Reward Models (PRMs), and problem difficulty influence TTS. This lack of analysis limits the understanding and practical use of TTS methods. In this paper, we focus on two core questions: **(1)** What is the optimal approach to scale test-time computation across different policy models, PRMs, and problem difficulty levels? **(2)** To what extent can extended computation improve the performance of LLMs on complex tasks, and can smaller language models outperform larger ones through this approach? Through comprehensive experiments on MATH-500 and challenging AIME24 tasks, we have the following observations: **(1)** The compute-optimal TTS strategy is highly dependent on the choice of policy model, PRM, and problem difficulty. **(2)** With our compute-optimal TTS strategy, extremely small policy models can outperform larger models. For example, a **1B** LLM can exceed a **405B** LLM on MATH-500. Moreover, on both MATH-500 and AIME24, a **0.5B** LLM outperforms **GPT-4o**, a **3B** LLM surpasses a **405B** LLM, and a **7B** LLM beats **o1** and **DeepSeek-R1**, while with **higher** inference efficiency. These findings show the significance of adapting TTS strategies to the specific characteristics of each task and model and indicate that TTS is a promising approach for enhancing the reasoning abilities of LLMs.

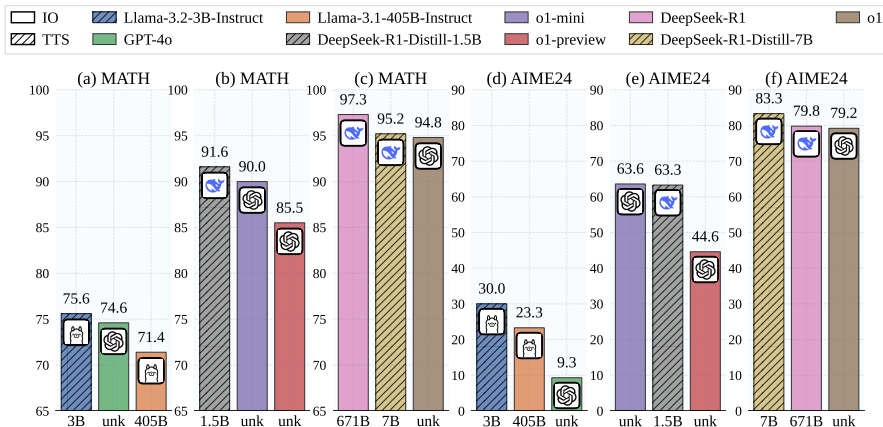


Figure 1: Comparison between the performance of large LLMs CoT and that of small LLMs compute-optimal TTS on MATH-500 and AIME24. **(a) & (d)** Llama-3.2-3B-Instruct surpasses Llama-3.1-405B-Instruct and GPT-4o on MATH-500 and AIME24; **(b) & (e)** DeepSeek-R1-Distill-1.5B outperforms o1-preview on MATH-500 and AIME24, and surpasses o1-mini on MATH-500; **(c) & (f)** DeepSeek-R1-Distill-7B beats o1 on MATH-500 and AIME24, and exceeds DeepSeek-R1 on AIME24.

## 1 INTRODUCTION

Large Language Models (LLMs) have shown significant improvements across a variety of domains (OpenAI, 2023; Hurst et al., 2024; Anthropic, 2023; OpenAI, 2024; DeepSeek-AI et al., 2025). Recently, OpenAI o1 (OpenAI, 2024) has demonstrated that Test-Time Scaling (TTS) can enhance the reasoning capabilities of LLMs by allocating additional computation at inference time, making it an effective approach for improving LLM performance (Qwen Team, 2024; Kimi Team et al., 2025; DeepSeek-AI et al., 2025).

The key challenge of TTS is how to scale compute optimally, that is, allocating the optimal computation for each problem (Snell et al., 2024). Current TTS methods guide the generation process and select the final answer using Process Reward Models (PRMs), which effectively scale test-time compute (Wu et al., 2024; Snell et al., 2024; Beeching et al., 2024). These TTS methods involve several important factors, such as policy models<sup>1</sup>, PRMs, and problem difficulty levels. However, there is limited systematic analysis of how policy models, PRMs, and problem difficulty influence these TTS strategies. This limitation prevents the community from fully understanding the effectiveness of this method and developing insights for compute-optimal TTS strategies.

To address these issues, this paper aims to investigate the influence of policy models, PRMs, and problem difficulty on TTS through comprehensive experimental analysis. Furthermore, we explore the concrete characteristics and performance boundaries of TTS methods. Specifically, we conduct extensive experiments on MATH-500 (Lightman et al., 2024) and the challenging AIME24 (AI-MO, 2024) tasks using a range of PRMs (spanning from 1.5B to 72B across different model series) across multiple policy models (ranging from 0.5B to 72B across two model families). Our results show that the compute-optimal TTS strategy heavily depends on the specific policy model, PRM, and problem difficulty level. Even smaller models (e.g., a **1B** model) can outperform larger models (e.g., a **405B** model) and even state-of-the-art reasoning models, such as **o1** or **DeepSeek-R1**, in challenging reasoning tasks by applying compute-optimal TTS.

The contributions of this work can be summarized as follows:

1. We conduct a comprehensive evaluation of different TTS methods using various up-to-date policy models, multiple PRMs, diverse scaling methods, and more challenging tasks.
2. Our analysis highlights the necessity of considering the influence of rewards in the TTS process and introduces reward-aware compute-optimal TTS. We also demonstrate that the compute-optimal scaling strategy varies with different policy models, PRMs, and problem difficulty levels.
3. The empirical results demonstrate the significant potential of smaller language models to outperform larger models through TTS. Using the reward-aware Compute-optimal TTS strategy, we show that a **3B** LLM can outperform a **405B** LLM, and a **7B** LLM can surpass **o1** and **DeepSeek-R1** on MATH-500 and AIME24 tasks.

## 2 SETUP & PRELIMINARIES

### 2.1 PROBLEM FORMULATION

We formulate the reasoning problem as a Markov Decision Process (MDP) (Sutton & Barto, 2018), defined by the tuple  $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$ , where  $\mathcal{S}$  is the state space,  $\mathcal{A}$  is the action space,  $\mathcal{P} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$  is the transition function,  $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$  is the reward function, and  $\gamma \in [0, 1]$  is the discount factor. Given a prompt  $x \sim \mathcal{X}$ , the policy with parameters  $\theta$  generates the initial action  $a_1 \sim \pi_\theta(\cdot | s_1)$ , where  $s_1 = x$  is the initial state. The policy receives a reward  $\mathcal{R}(s_1, a_1)$ , and the state transitions to  $s_2 = [s_1, a_1]$ , where  $[\cdot, \cdot]$  denotes the concatenation of two strings. This process continues until the episode terminates, either by reaching the maximum number of steps or by generating an `<EOS>` token. A trajectory of length  $H$  is represented as  $\tau = \{a_1, a_2, \dots, a_H\}$ . The process can be summarized as follows:

$$\begin{array}{ll}
 \text{Initial State:} & s_1 = x \sim \mathcal{X} \\
 \text{Action:} & a_t \sim \pi_\theta(\cdot | s_t) \\
 \text{State Transition:} & s_{t+1} = \mathcal{P}(\cdot | s_t, a_t) = [s_t, a_t] \\
 \text{Reward:} & r_t = \mathcal{R}(s_t, a_t)
 \end{array} \tag{1}$$

<sup>1</sup>Following Snell et al. (2024), we use “policy models” to refer to LLMs that generate solutions, and “verifiers” for PRMs.

## 2.2 TEST-TIME SCALING METHOD

We consider three TTS methods: Best-of-N (BoN) (Brown et al., 2024), beam search (Snell et al., 2024), and Diverse Verifier Tree Search (DVTS) (Beeching et al., 2024). As pointed out by Snell et al. (2024), lookahead search is inefficient due to multi-step sampling, so we do not evaluate it or other methods involving lookahead operations, such as Monte Carlo Tree Search (MCTS).

**Best-of-N.** In the BoN approach, the policy model generates  $N$  responses, after which scoring and voting methods are applied to select the final answer.

**Beam Search.** Given beam width  $N$  and beam size  $M$ , the policy model first generates  $N$  steps. The verifier selects the top  $\frac{N}{M}$  steps for subsequent search. In the next step, the policy model samples  $M$  steps for each selected previous step. This process repeats until the maximum depth is reached or an  $\langle \text{EOS} \rangle$  token is generated.

**Diverse Verifier Tree Search.** To increase diversity, DVTS extends beam search by dividing the search process into  $\frac{N}{M}$  subtrees, each of which is explored independently using beam search. As shown in Beeching et al. (2024), DVTS outperforms beam search on easy and medium problems with a large computational budget  $N$ . A similar trend is observed in Chen et al. (2024), where increasing the number of parallel subtrees proves to be more effective than increasing the beam width under the same budget.

## 2.3 COMPUTE-OPTIMAL TEST-TIME SCALING

To maximize the performance of TTS, Snell et al. (2024) proposes a test-time compute-optimal scaling strategy, which selects hyperparameters corresponding to a given test-time strategy to maximize performance benefits on a specific prompt. Given a prompt  $x$ , let  $\text{Target}(\theta, N, x)$  represent the output distribution over  $x$  produced by the policy model with parameters  $\theta$  and a compute budget of  $N$ .

$$\theta_{x, y^*(x)}^*(N) = \arg \max_{\theta} (\mathbb{E}_{y \sim \text{Target}(\theta, N, x)} [\mathbb{1}_{y=y^*(x)}]), \quad (2)$$

where  $y^*(x)$  denotes the ground-truth correct response for  $x$ , and  $\theta_{x, y^*(x)}^*(N)$  represents the test-time compute-optimal scaling strategy for the problem  $x$  with compute budget  $N$ .

## 3 RETHINKING COMPUTE-OPTIMAL TEST-TIME SCALING

### 3.1 COMPUTE-OPTIMAL SCALING STRATEGY SHOULD BE REWARD-AWARE

Compute-optimal TTS aims to allocate the optimal compute for each problem (Snell et al., 2024). Previous works on TTS use a single PRM as verifier (Snell et al., 2024; Wu et al., 2024; Beeching et al., 2024). Snell et al. (2024) trains a PRM on the responses of a policy model and uses it as the verifier to do TTS with the same policy model, while Wu et al. (2024); Beeching et al. (2024) use a PRM trained on a different policy model to do TTS. From the perspective of reinforcement learning, we obtain an *on-policy* PRM in the former case and an *offline* PRM in the latter case. The on-policy PRM produces more accurate rewards for the responses of the policy model, while the offline PRM often generates inaccurate rewards due to out-of-distribution (OOD) issues (Snell et al., 2024; Zheng et al., 2024).

For practical applications of compute-optimal TTS, training a PRM for each policy model to prevent OOD issues is computationally expensive. Therefore, we investigate the compute-optimal TTS strategy in a more general setting, where the PRM might be trained on a different policy model than the one used for TTS. For search-based methods, PRMs guide the selection at each response step, while for sampling-based methods, PRMs evaluate the responses after generation. This indicates that (1) the reward influences response selection across all methods; (2) for search-based methods, the reward also influences the search process.

To analyze these points, we perform a preliminary case study using beam search with Llama-3.1-8B-Instruct as the policy model and RLHFlow-PRM-Mistral-8B and RLHFlow-PRM-Deepseek-8B as PRMs. The results in Figure 11 demonstrate that the reward significantly affects the generation process and outcomes. RLHFlow-PRM-Mistral-8B assigns high rewards to short responses, leading to incorrect answers, while searching with RLHFlow-Deepseek-PRM-8B produces correct answers but uses more tokens. In Section 4, we also empirically show that rewards have great influence on TTS performance and output tokens.

Based on these findings, we propose that *rewards* should be integrated into the compute-optimal TTS strategy. Let us denote the reward function as  $\mathcal{R}$ . Our reward-aware compute-optimal TTS strategy is formulated as:

$$\theta_{x,y^*(x),\mathcal{R}}^*(N) = \arg \max_{\theta} \left( \mathbb{E}_{y \sim \text{Target}(\theta, N, x, \mathcal{R})} [\mathbb{1}_{y=y^*(x)}] \right), \quad (3)$$

where  $\text{Target}(\theta, N, x, \mathcal{R})$  represents the output distribution of the policy model  $\theta$ , adjusted by the reward function  $\mathcal{R}$ , under a compute budget  $N$  and prompt  $x$ . For sampling-based scaling methods,  $\text{Target}(\theta, N, x, \mathcal{R}) = \text{Target}(\theta, N, x)$ . This reward-aware strategy ensures that compute-optimal scaling adapts to the policy model, prompt, and reward function, leading to a more general framework for practical TTS.

### 3.2 ABSOLUTE PROBLEM DIFFICULTY CRITERION IS MORE EFFECTIVE THAN QUANTILES

To consider the influence of problem difficulty on TTS, Snell et al. (2024) group problems into five difficulty levels based on Pass@1 accuracy quantiles. However, we find that using difficulty levels from MATH (Hendrycks et al., 2021) or oracle labels based on Pass@1 accuracy quantiles (Snell et al., 2024) is not effective since different policy models have different reasoning capabilities. As shown in Figure 2, Qwen2.5-72B-Instruct achieves Pass@1 accuracy above 80% on 76.2% of MATH-500 problems. Therefore, we use absolute thresholds instead of quantiles to measure problem difficulty. Specifically, we define three difficulty levels based on Pass@1 accuracy: easy (50% ~ 100%), medium (10% ~ 50%), and hard (0% ~ 10%).

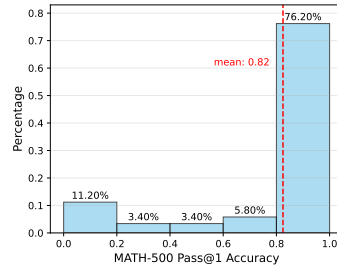


Figure 2: Distribution of Pass@1 accuracy of Qwen2.5-72B-Instruct on MATH-500, divided into five bins.

## 4 HOW TO SCALE TEST-TIME COMPUTE OPTIMALLY?

In this section, we aim to answer the following questions:

- **Q1:** How does TTS improve with different policy models and PRMs?
- **Q2:** How does TTS improve for problems with different difficulty levels?

### 4.1 SETUP

**Datasets.** We conduct experiments on competition-level mathematical datasets, including MATH-500 (Lightman et al., 2024) and AIME24 (AI-MO, 2024). MATH-500 contains 500 representative problems from the test set of MATH (Hendrycks et al., 2021), and this subset is used following Snell et al. (2024); Beeching et al. (2024). As recent LLMs show significant progress in mathematical reasoning (OpenAI, 2024; DeepSeek-AI et al., 2025), we include the more challenging AIME24 for experiments.

**Policy Models.** For test-time methods, we use policy models from Llama 3 (Dubey et al., 2024) and Qwen2.5 (Yang et al., 2024b) families with different sizes. We use the *Instruct* version for all policy models.

**Process Reward Models.** We consider the following open-source PRMs for evaluation:

- **Math-Shepherd** (Wang et al., 2024b): **Math-Shepherd-PRM-7B** is trained on Mistral-7B (Jiang et al., 2023) using PRM data generated from Mistral-7B fine-tuned on MetaMath (Yu et al., 2024).
- **RLHFlow Series** (Xiong et al., 2024): RLHFlow includes **RLHFlow-PRM-Mistral-8B** and **RLHFlow-PRM-Deepseek-8B**, which are trained on data from Mistral-7B fine-tuned on MetaMath (Yu et al., 2024) and deepseek-math-7b-instruct (Shao et al., 2024), respectively. The base model for both PRMs is Llama-3.1-8B-Instruct (Dubey et al., 2024).
- **Skywork Series** (Skywork o1 Team, 2024): The Skywork series comprises **Skywork-PRM-1.5B** and **Skywork-PRM-7B**, trained on Qwen2.5-Math-1.5B-Instruct and Qwen2.5-Math-7B-Instruct (Yang et al., 2024c), respectively. The training data is generated from Llama-2 (Touvron

216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269

et al., 2023) fine-tuned on a mathematical dataset and Qwen2-Math (Yang et al., 2024a) series models.

- **Qwen2.5-Math Series** (Zhang et al., 2025): We evaluate **Qwen2.5-Math-PRM-7B** and **Qwen2.5-Math-PRM-72B**, trained on Qwen2.5-Math-7B-Instruct and Qwen2.5-Math-72B-Instruct (Yang et al., 2024c), respectively. The data for training is generated using Qwen2-Math (Yang et al., 2024a) and Qwen2.5-Math series models (Yang et al., 2024c). Among all the PRMs listed, Qwen2.5-Math-PRM-72B is the strongest open-source PRM for mathematical tasks, while Qwen2.5-Math-PRM-7B is the most capable PRM among those with 7B/8B parameters, as demonstrated in Zhang et al. (2025).

4.2 HOW DOES TTS IMPROVE WITH DIFFERENT POLICY MODELS AND PRMS? (Q1)

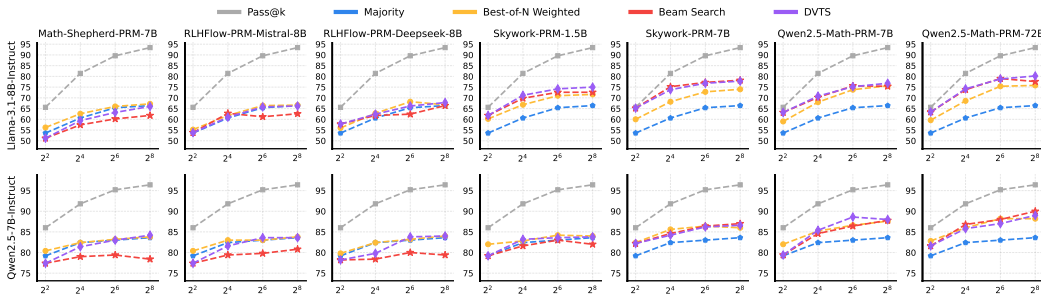


Figure 3: Performance of Llama-3.1-8B-Instruct and Qwen2.5-7B-Instruct on MATH-500 with different PRMs and TTS strategies.

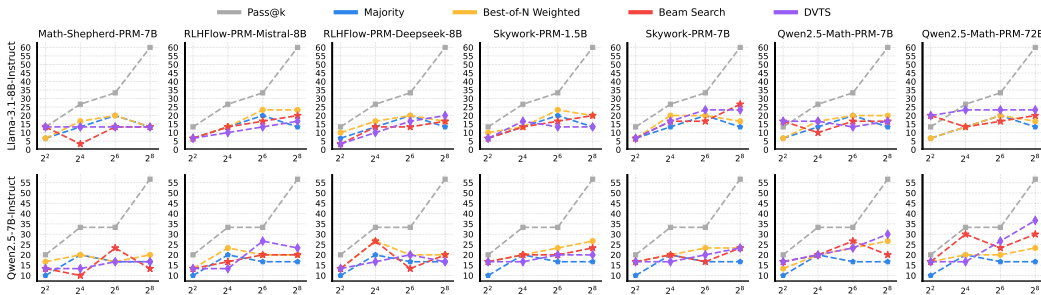


Figure 4: Performance of Llama-3.1-8B-Instruct and Qwen2.5-7B-Instruct on AIME24 with different PRMs and TTS strategies.

**PRMs are hard to generalize across policy models and tasks.** As shown in Figure 3, for Llama-3.1-8B-Instruct, the performance of search-based methods with Skywork and Qwen2.5-Math PRMs improves significantly with larger compute budgets, while the results of searching with Math-Shepherd and RLHFlow PRMs remain relatively poor, even worse than majority voting. For Qwen2.5-7B-Instruct, the performance of searching with Skywork-PRM-7B and Qwen2.5-Math PRMs scales well with more budgets, while the performance of other PRMs remains poor. In Figure 4, although the Pass@k accuracy of both policy models improves a lot with larger compute budgets, the performance improvement of TTS remains moderate. These results demonstrate that the generalization of PRMs is particularly challenging across different policy models and tasks, especially for more complex tasks.

**The optimal TTS method depends on the PRM used.** As shown in Figure 3, BoN outperforms other strategies most of the time when using Math-Shepherd and RLHFlow PRMs, while search-based methods perform better with Skywork and Qwen2.5-Math PRMs. This difference occurs because using a PRM for OOD policy responses leads to sub-optimal answers, as PRMs show limited generalization across policy models. Moreover, if we select *each step* with OOD PRMs, it is likely to obtain answers trapped in local optima and worsen the performance. This may also be related to the base model of the PRM, since the PRM trained with PRM800K (Lightman et al., 2024) on Qwen2.5-Math-7B-Instruct generalizes better than PRMs with Mistral and Llama as base models (Zhang et al.,

2025). Further analysis is provided in Section D.1 and Appendix F. These results suggest that the choice of the optimal TTS strategy depends on the specific PRMs used, emphasizing the importance of considering reward information in compute-optimal TTS. We also explore the relationship between TTS performance and the process supervision abilities of different PRMs. As shown in Figure 5, TTS performance is positively correlated with the process supervision abilities of PRMs, and the fitted function is  $Y = 7.66 \log(X) + 44.31$ , where  $Y$  represents TTS performance and  $X$  represents the process supervision abilities of the PRM (Zhang et al., 2025).

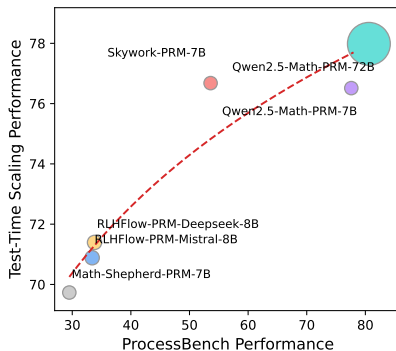


Figure 5: The relationship between TTS performance and process supervision abilities of different PRMs on MATH, where the size of each circle represents the number of parameters of the PRM and the curve represents the fitted function.

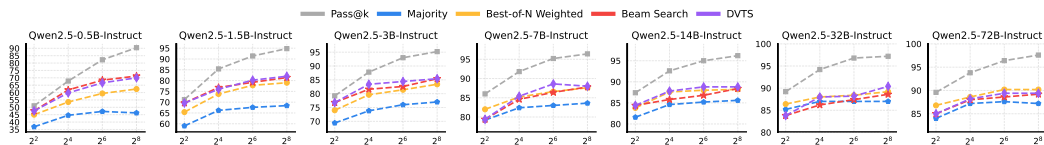


Figure 6: TTS performance of policy models with parameters from 0.5B to 72B on MATH-500 with different scaling methods.

**The optimal TTS method varies with policy models.** To study the relationship between the parameters of the policy models and the optimal TTS methods, we conduct experiments with Qwen2.5 family LLMs (Yang et al., 2024b), including models with 0.5B, 1.5B, 3B, 7B, 14B, 32B, and 72B parameters. The results in Figure 6 show that the optimal TTS methods depend on the specific policy models. For small policy models, search-based methods outperform BoN, while for large policy models, BoN is more effective than search-based methods. This difference occurs because larger models have stronger reasoning capabilities and do not need a verifier to perform step-by-step selection. In contrast, smaller models rely on a verifier to select each step, ensuring the correctness of each intermediate step.

#### 4.3 HOW DOES TTS IMPROVE FOR PROBLEMS WITH DIFFERENT DIFFICULTY LEVELS? (Q2)

Following Snell et al. (2024), we conduct a comprehensive evaluation of tasks with varying difficulty levels. However, as explained in Section 3.2, we observe that using the difficulty levels defined in MATH (Hendrycks et al., 2021) or the oracle labels based on the quantile of Pass@1 accuracy (Snell et al., 2024) is not appropriate because different policy models exhibit different reasoning abilities. To address this, we categorize the difficulty levels into three groups based on the absolute value of Pass@1 accuracy: easy (50% ~ 100%), medium (10% ~ 50%), and hard (0% ~ 10%).

**The optimal TTS methods vary with different difficulty levels.** The results in Figure 7 and Figure 8 show that for small policy models (i.e., with fewer than 7B parameters), BoN is better for easy problems, while beam search works better for harder problems. For policy models with parameters between 7B and 32B, DVTS performs well for easy and medium problems, and beam search is preferable for hard problems. For policy models with 72B parameters, BoN is the best method for all difficulty levels.

324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377

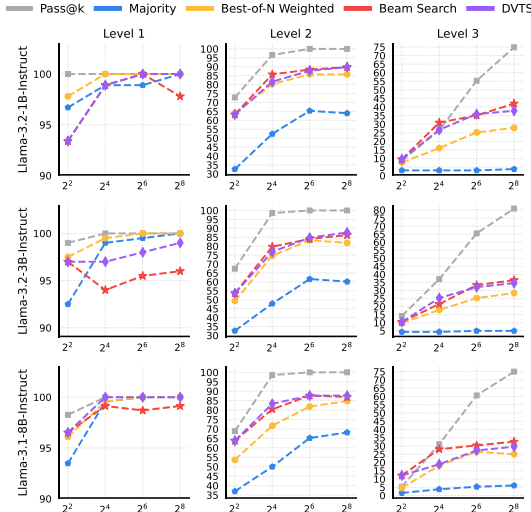


Figure 7: TTS performance of three Llama policy models on MATH-500 with three difficulty levels.

## 5 RESULTS FOR COMPUTE-OPTIMAL TEST-TIME SCALING

With the compute-optimal TTS strategy explored in Section 4, we conduct further experiments to explore the following questions:

- **Q3:** Can smaller policy models outperform larger models with the compute-optimal TTS strategy?
- **Q4:** How does compute-optimal TTS improve compared with CoT and majority voting?
- **Q5:** Is TTS more effective than long-CoT-based methods?

### 5.1 CAN SMALLER POLICY MODELS OUTPERFORM LARGER MODELS WITH THE COMPUTE-OPTIMAL TTS STRATEGY (Q3)

Scaling test-time compute of small policy models is crucially important for improving the reasoning performance of LLMs. We are interested in whether smaller policy models can outperform larger ones, GPT-4o, even o1 and DeepSeek-R1, with the compute-optimal TTS strategy. First, we compare the performance of Llama-3.2-3B-Instruct (compute-optimal TTS) with that of Llama-3.1-405B-Instruct (CoT) on MATH-500 and AIME24. Also, we compare the performance of Qwen2.5-0.5B-Instruct, Qwen2.5-1.5B-Instruct, Llama-3.2-1B-Instruct, and Llama-3.2-3B-Instruct with GPT-4o on the above two tasks. As AIME24 is challenging for current LLMs, we also compare the performance of DeepSeek-R1-Distill-Qwen-1.5B and DeepSeek-R1-Distill-Qwen-7B with o1 on AIME24.

From the results in Table 1, we have the following observations: (1) Llama-3.2-3B-Instruct with the compute-optimal TTS strategy outperforms Llama-3.1-405B-Instruct on MATH-500 and AIME24, meaning that **smaller models can outperform 135× larger models using the compute-optimal TTS strategy**. Compared with previous works on TTS (Snell et al., 2024; Beeching et al., 2024), we improve the result by **487.0%** ( $23\times \rightarrow 135\times$ ). (2) If we further increase the compute budget to  $N = 512$ , Llama-3.2-1B-Instruct with the compute-optimal TTS strategy beats Llama-3.1-405B-Instruct on MATH-500, but underperforms Llama-3.1-405B-Instruct on AIME24.<sup>2</sup> (3) Qwen2.5-0.5B-Instruct and Llama-3.2-3B-Instruct with the compute-optimal TTS strategy outperforms GPT-4o, indicating that **small models can exceed GPT-level performance with the compute-optimal TTS strategy**. (4) DeepSeek-R1-Distill-Qwen-1.5B with the compute-optimal TTS strategy outperforms o1-preview and o1-mini on MATH-500 and AIME24. We also show that DeepSeek-R1-Distill-Qwen-7B with the compute-optimal TTS strategy outperforms o1 and DeepSeek-R1 on MATH-500 and AIME24. These results demonstrate **small reasoning-enhanced models can outperform frontier reasoning LLMs with the compute-optimal TTS strategy**.

<sup>2</sup>Since some outputs of Llama-3.2-1B-Instruct do not contain `\boxed`, which is used for answer extraction, we use Qwen2.5-32B-Instruct to extract the answers of Llama-3.2-1B-Instruct.

Table 1: Comparison of small policy models (compute-optimal TTS) with frontier reasoning LLMs (CoT) on MATH-500 and AIME24.

Policy Model	MATH-500	AIME24	Avg.
<i>Proprietary LLMs</i>			
GPT-4o	74.6	9.3	42.0
o1-preview	85.5	44.6	65.1
o1-mini	90.0	63.6	76.8
o1	94.8	79.2	87.0
<i>Open-Source LLMs</i>			
Llama-3.1-70B-Inst.	65.2	16.7	41.0
Llama-3.1-405B-Inst.	71.4	23.3	47.4
QwQ-32B-Preview	90.6	50.0	70.3
DeepSeek-R1	97.3	79.8	88.6
<i>Open-Source LLMs with TTS</i>			
Llama-3.2-1B-Inst.	66.2	16.7	41.5
Llama-3.2-1B-Inst. ( $N = 512$ )	72.2	10.0	41.1
Llama-3.2-3B-Inst.	75.6	30.0	52.8
Qwen2.5-0.5B-Inst.	76.4	10.0	43.2
Qwen2.5-1.5B-Inst.	81.8	20.0	50.9
<i>Open-Source Reasoning-Enhanced LLMs with TTS</i>			
DeepSeek-R1-Distill-Qwen-1.5B	91.6	63.3	77.5
DeepSeek-R1-Distill-Qwen-7B	95.2	83.3	89.3

Table 2: FLOPS Comparison between smaller policy models (compute-optimal TTS) and larger ones (CoT).

Policy Model	Pre-training FLOPS	Inference FLOPS	Total FLOPS.
Llama-3.2-3B-Inst.	$1.62 \times 10^{23}$	$3.00 \times 10^{17}$	$1.62 \times 10^{23}$
Llama-3.1-405B-Inst.	$3.15 \times 10^{26}$	$4.05 \times 10^{17}$	$3.15 \times 10^{26}$
DeepSeek-R1-Distill-7B	$7.56 \times 10^{23}$	$3.00 \times 10^{17}$	$7.56 \times 10^{23}$
DeepSeek-R1	$5.96 \times 10^{25}$	$6.71 \times 10^{17}$	$5.96 \times 10^{25}$

**FLOPS Comparison.** To answer the question of whether compute-optimal TTS is more effective than increasing the model size, we compare the FLOPS of evaluated models in Table 2 following Snell et al. (2024), where the computed FLOPS is corresponded to the results in Table 1. From the results, we can see that **small policy models even surpass large ones with less inference FLOPS** and reduce the total FLOPS by  $100\times \sim 1000\times$ .

## 5.2 HOW DOES COMPUTE-OPTIMAL TTS IMPROVE COMPARED WITH CoT AND MAJORITY VOTING? (Q4)

Based on the findings of compute-optimal TTS with different policy models, PRMs, and difficulty levels, we summarize the results of compute-optimal TTS for each policy model on MATH-500 in Table 3. We find that compute-optimal TTS can be  $60\times$  more efficient than majority voting and improve reasoning performance by 150% over CoT. These results demonstrate that compute-optimal TTS significantly enhances the reasoning capabilities of LLMs. However, as the number of parameters in the policy model increases, the improvement of TTS gradually decreases. This suggests that the effectiveness of TTS is directly related to the reasoning ability of the policy model. Specifically, for models with weak reasoning abilities, scaling test-time compute leads to a substantial improvement, whereas for models with strong reasoning abilities, the gain is limited.

## 5.3 IS TTS MORE EFFECTIVE THAN LONG-CoT-BASED METHODS? (Q5)

Recently, long-CoT-based methods have shown substantial progress in mathematical reasoning (Guan et al., 2025; Cui et al., 2025; Zeng et al., 2025; DeepSeek-AI et al., 2025). We compare the performance of TTS with these approaches.

**Setup.** We evaluate the following methods: (1) **rStar-Math** (Guan et al., 2025): This method first generates reasoning data via MCTS, followed by online policy and preference model learning. (2)



Table 3: Comparison of TTS with different policy models on MATH-500.

Policy Model	CoT	Major.	Compute-Optimal TTS	Performance Gain	Efficiency Gain
Llama-3.2-1B-Inst.	26.0	39.0	65.6	152.3%	60.0×
Llama-3.2-3B-Inst.	41.4	58.4	77.6	87.4%	14.1×
Llama-3.1-8B-Inst.	49.8	66.4	80.2	61.0%	54.2×
Qwen2.5-0.5B-Inst.	31.6	47.2	76.0	140.5%	16.0×
Qwen2.5-1.5B-Inst.	54.4	68.4	82.6	51.8%	46.7×
Qwen2.5-3B-Inst.	64.0	77.0	87.4	36.6%	64.0×
Qwen2.5-7B-Inst.	76.8	83.6	90.0	17.2%	45.3×
Qwen2.5-14B-Inst.	80.2	85.6	90.8	13.2%	16.0×
Qwen2.5-32B-Inst.	82.4	87.0	90.4	9.7%	4.0×
Qwen2.5-72B-Inst.	83.8	87.2	90.2	7.6%	4.0×

Table 4: Comparison of compute-optimal TTS with long-CoT methods on MATH-500 and AIME24.

Policy Model	MATH-500	AIME24	Avg.
<i>Open-Source Models with Direct CoT</i>			
Qwen2.5-7B-Inst.	76.8	13.3	45.1
Qwen2.5-Math-7B-Inst.	79.8	13.3	46.6
<i>Long-CoT Methods with Direct CoT</i>			
rStar-Math-7B	78.4	26.7	52.6
Eurus-2-7B-PRIME	79.2	26.7	53.0
Qwen2.5-7B-SimpleRL-Zero	77.2	33.3	55.3
Qwen2.5-7B-SimpleRL	82.4	26.7	54.6
DeepSeek-R1-Distill-Qwen-7B	92.4	63.3	77.9
<i>Open-Source Models with TTS</i>			
Qwen2.5-7B-Inst. w/ 7B PRM (Ours)	88.0	33.3	60.5
Qwen2.5-7B-Inst. w/ 72B PRM (Ours)	91.0	36.7	63.9

**Eurus-2** (Cui et al., 2025): This method enhances the reasoning abilities of LLMs through implicit process rewards and online reinforcement learning. (3) **SimpleRL** (Zeng et al., 2025): This method replicates self-reflection with only 8K training data. (4) **DeepSeek-R1-Distill-Qwen-7B** (DeepSeek-AI et al., 2025): This method distills 800K high-quality reasoning samples from DeepSeek-R1 with 671B parameters into a 7B LLM.

**Results.** As shown in Table 4, we find that TTS with Qwen2.5-7B-Instruct outperforms rStar-Math, Eurus-2, and SimpleRL on both MATH-500 and AIME24. However, while the performance of TTS on MATH-500 is close to that of DeepSeek-R1-Distill-Qwen-7B, it shows a significant drop on AIME24. These results indicate that TTS is more effective than methods applying direct reinforcement learning or SFT on the data generated via MCTS but is less effective than distilling from strong reasoning models.

## 6 CONCLUSION & DISCUSSION

In this paper, we conduct a detailed empirical study on compute-optimal test-time scaling from the perspectives of different policy models, PRMs, and more challenging evaluation tasks. Our results demonstrate the dependency of optimal TTS strategies on policy models and validate the significant potential of small language models to outperform large-scale models through TTS. For instance, a mere 1B size model can surpass a 405B model with TTS. We also show that a 7B PRM can achieve stronger TTS by supervising a more powerful 72B policy model, highlighting the need to explore a true “weak-to-strong” paradigm beyond the current “strong-to-weak” supervision for policy optimization. The key to this goal is designing more efficient forms of supervision, as PRM-based and reinforcement learning-based methods still have limitations due to their reliance on high-quality supervision. Future work should investigate more flexible and general supervision mechanisms to boost the performance of small language models in complex tasks and provide new directions for developing resource-efficient reasoning strategies.

**Limitations.** Although we provide a comprehensive evaluation of TTS on mathematical tasks, there are still some limitations and future directions to explore: (1) Extending TTS to more tasks such as coding and chemistry tasks. (2) Exploring more effective methods for compute-optimal TTS.

## REFERENCES

- 486  
487  
488 AI-MO. Aime 2024, 2024. URL [https://huggingface.co/datasets/AI-MO/](https://huggingface.co/datasets/AI-MO/aimo-validation-aime)  
489 [aimo-validation-aime](https://huggingface.co/datasets/AI-MO/aimo-validation-aime).
- 490 Anthropic. Introducing Claude, 2023. URL [https://www.anthropic.com/index/](https://www.anthropic.com/index/introducing-claude/)  
491 [introducing-claude/](https://www.anthropic.com/index/introducing-claude/).
- 492  
493 Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen Marcus McAleer,  
494 Albert Q. Jiang, Jia Deng, Stella Biderman, and Sean Welleck. Llemma: An open language model  
495 for mathematics. In *International Conference on Learning Representations (ICLR)*, 2024. URL  
496 <https://openreview.net/forum?id=4WnqRR915j>.
- 497 Edward Beeching, Lewis Tunstall, and Sasha Rush. Scaling test-time compute with  
498 open models, 2024. URL [https://huggingface.co/spaces/HuggingFaceH4/](https://huggingface.co/spaces/HuggingFaceH4/blogpost-scaling-test-time-compute)  
499 [blogpost-scaling-test-time-compute](https://huggingface.co/spaces/HuggingFaceH4/blogpost-scaling-test-time-compute).
- 500  
501 Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and  
502 Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling.  
503 *arXiv preprint arXiv:2407.21787*, 2024.
- 504 Guoxin Chen, Minpeng Liao, Chengxi Li, and Kai Fan. Alphamath almost zero: Process supervision  
505 without process. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. URL  
506 <https://openreview.net/forum?id=VaXnxQ3UKo>.
- 507  
508 Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. Program of thoughts prompting:  
509 Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine*  
510 *Learning Research (TMLR)*, 2023. ISSN 2835-8856. URL [https://openreview.net/](https://openreview.net/forum?id=YfZ4ZPt8zd)  
511 [forum?id=YfZ4ZPt8zd](https://openreview.net/forum?id=YfZ4ZPt8zd).
- 512  
513 Ganqu Cui, Lifan Yuan, Zefan Wang, Hanbin Wang, Wendi Li, Bingxiang He, Yuchen Fan, Tianyu  
514 Yu, Qixin Xu, Weize Chen, Jiarui Yuan, Huayu Chen, Kaiyan Zhang, Xingtai Lv, Shuo Wang,  
515 Yuan Yao, Xu Han, Hao Peng, Yu Cheng, Zhiyuan Liu, Maosong Sun, Bowen Zhou, and Ning  
516 Ding. Process reinforcement through implicit rewards. *arXiv preprint arXiv:2502.01456*, 2025.
- 517  
518 DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu,  
519 Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu,  
520 Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao  
521 Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan,  
522 Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao,  
523 Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding,  
524 Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang  
525 Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong,  
526 Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao,  
527 Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang,  
528 Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang,  
529 Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L.  
530 Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang,  
531 Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng  
532 Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanbiao Zhao, Wen Liu, Wenfeng  
533 Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan  
534 Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang,  
535 Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen,  
536 Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li,  
537 Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang,  
538 Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan,  
539 Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia  
540 He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong  
541 Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha,  
542 Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang,  
543 Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li,  
544 Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen

- 540 Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv*  
541 *preprint arXiv:2501.12948*, 2025.
- 542
- 543 Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha  
544 Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models.  
545 *arXiv preprint arXiv:2407.21783*, 2024.
- 546
- 547 Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and  
548 Graham Neubig. PAL: Program-aided language models. In *International Conference on Machine*  
549 *Learning (ICML)*, volume 202, pp. 10764–10799, 2023.
- 550 Zhibin Gou, Zhihong Shao, Yeyun Gong, yelong shen, Yujiu Yang, Minlie Huang, Nan Duan,  
551 and Weizhu Chen. ToRA: A tool-integrated reasoning agent for mathematical problem solv-  
552 ing. In *International Conference on Learning Representations (ICLR)*, 2024. URL <https://openreview.net/forum?id=Ep0TtjVoap>.
- 553
- 554 Xinyu Guan, Li Lyna Zhang, Yifei Liu, Ning Shang, Youran Sun, Yi Zhu, Fan Yang, and Mao Yang.  
555 rStar-Math: Small llms can master math reasoning with self-evolved deep thinking. *arXiv preprint*  
556 *arXiv:2501.04519*, 2025.
- 557
- 558 Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Ksenia Konyushkova, Lotte Weerts, Abhishek  
559 Sharma, Aditya Siddhant, Alex Ahern, Miaosen Wang, Chenjie Gu, et al. Reinforced self-training  
560 (rest) for language modeling. *arXiv preprint arXiv:2308.08998*, 2023.
- 561
- 562 Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song,  
563 and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In  
564 *Advances in Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*,  
565 2021. URL <https://openreview.net/forum?id=7Bywt2mQsCe>.
- 566
- 567 Arian Hosseini, Xingdi Yuan, Nikolay Malkin, Aaron Courville, Alessandro Sordoni, and Rishabh  
568 Agarwal. V-star: Training verifiers for self-taught reasoners. *arXiv preprint arXiv:2402.06457*,  
569 2024.
- 570
- 571 Zhen Huang, Haoyang Zou, Xuefeng Li, Yixiu Liu, Yuxiang Zheng, Ethan Chern, Shijie Xia, Yiwei  
572 Qin, Weizhe Yuan, and Pengfei Liu. O1 replication journey—part 2: Surpassing o1-preview through  
573 simple distillation, big progress or bitter lesson? *arXiv preprint arXiv:2411.16489*, 2024.
- 574
- 575 Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Os-  
576 trow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint*  
577 *arXiv:2410.21276*, 2024.
- 578
- 579 Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot,  
580 Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al.  
581 Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.
- 582
- 583 Jikun Kang, Xin Zhe Li, Xi Chen, Amirreza Kazemi, Qianyi Sun, Boxing Chen, Dong Li, Xu He,  
584 Quan He, Feng Wen, et al. MindStar: Enhancing math reasoning in pre-trained llms at inference  
585 time. *arXiv preprint arXiv:2405.16265*, 2024.
- 586
- 587 Maxim Khanov, Jirayu Burapachee, and Yixuan Li. ARGS: Alignment as reward-guided  
588 search. In *International Conference on Learning Representations (ICLR)*, 2024. URL <https://openreview.net/forum?id=shgx0eqdw6>.
- 589
- 590 Kimi. k0-math, November 2024. URL <https://kimi.moonshot.cn/>.
- 591
- 592 Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun  
593 Xiao, Chenzhuang Du, Chonghua Liao, et al. Kimi k1.5: Scaling reinforcement learning with llms.  
*arXiv preprint arXiv:2501.12599*, 2025.
- 594
- 595 Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli,  
596 Shariq Iqbal, Colton Bishop, Rebecca Roelofs, et al. Training language models to self-correct via  
597 reinforcement learning. *arXiv preprint arXiv:2409.12917*, 2024.

- 594 Joshua Ong Jun Leang, Aryo Pradipta Gema, and Shay B Cohen. CoMAT: Chain of mathematically  
595 annotated thought improves mathematical reasoning. *arXiv preprint arXiv:2410.10336*, 2024.  
596
- 597 Wendi Li and Yixuan Li. Process reward model with q-value rankings. *arXiv preprint*  
598 *arXiv:2410.11287*, 2024.  
599
- 600 Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan  
601 Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let’s verify step by step. In *International*  
602 *Conference on Learning Representations (ICLR)*, 2024. URL [https://openreview.net/](https://openreview.net/forum?id=v8L0pN6EOi)  
603 [forum?id=v8L0pN6EOi](https://openreview.net/forum?id=v8L0pN6EOi).
- 604 Jianqiao Lu, Zhiyang Dou, WANG Hongru, Zeyu Cao, Jianbo Dai, Yunlong Feng, and Zhijiang Guo.  
605 Autopsv: Automated process-supervised verifier. In *Advances in Neural Information Processing*  
606 *Systems (NeurIPS)*, 2024.
- 607 Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Jianguang Lou, Chongyang Tao, Xiubo Geng,  
608 Qingwei Lin, Shifeng Chen, and Dongmei Zhang. Wizardmath: Empowering mathematical  
609 reasoning for large language models via reinforced evol-instruct. *arXiv preprint arXiv:2308.09583*,  
610 2023.  
611
- 612 Liangchen Luo, Yinxiao Liu, Rosanne Liu, Samrat Phatale, Harsh Lara, Yunxuan Li, Lei Shu, Yun  
613 Zhu, Lei Meng, Jiao Sun, et al. Improve mathematical reasoning in language models by automated  
614 process supervision. *arXiv preprint arXiv:2406.06592*, 2024.
- 615 Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegrefe, Uri Alon,  
616 Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder,  
617 Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative  
618 refinement with self-feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*,  
619 volume 36, pp. 46534–46594, 2023.  
620
- 621 Rohin Manvi, Anikait Singh, and Stefano Ermon. Adaptive inference-time compute: Llms can predict  
622 if they can do better, even mid-generation. *arXiv preprint arXiv:2410.02725*, 2024.
- 623 OpenAI. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.  
624
- 625 OpenAI. Learning to reason with llms, 2024. URL [https://openai.com/index/](https://openai.com/index/learning-to-reason-with-llms/)  
626 [learning-to-reason-with-llms/](https://openai.com/index/learning-to-reason-with-llms/).
- 627 Yiwei Qin, Xuefeng Li, Haoyang Zou, Yixiu Liu, Shijie Xia, Zhen Huang, Yixin Ye, Weizhe Yuan,  
628 Hector Liu, Yuanzhi Li, et al. O1 replication journey: A strategic progress report–part 1. *arXiv*  
629 *preprint arXiv:2410.18982*, 2024.  
630
- 631 Yuxiao Qu, Tianjun Zhang, Naman Garg, and Aviral Kumar. Recursive introspection: Teaching  
632 language model agents how to self-improve. In *Advances in Neural Information Processing*  
633 *Systems (NeurIPS)*, 2024. URL <https://openreview.net/forum?id=DRC9pZwBwR>.  
634
- 635 Qwen Team. Qwq: Reflect deeply on the boundaries of the unknown, November 2024. URL  
636 <https://qwenlm.github.io/blog/qwq-32b-preview/>.
- 637 Amrith Setlur, Saurabh Garg, Xinyang Geng, Naman Garg, Virginia Smith, and Aviral Kumar. R1 on  
638 incorrect synthetic data scales the efficiency of llm math reasoning by eight-fold. *arXiv preprint*  
639 *arXiv:2406.14532*, 2024a.  
640
- 641 Amrith Setlur, Chirag Nagpal, Adam Fisch, Xinyang Geng, Jacob Eisenstein, Rishabh Agarwal,  
642 Alekh Agarwal, Jonathan Berant, and Aviral Kumar. Rewarding progress: Scaling automated  
643 process verifiers for llm reasoning. *arXiv preprint arXiv:2410.08146*, 2024b.
- 644 Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang,  
645 Mingchuan Zhang, YK Li, Y Wu, et al. DeepSeekMath: Pushing the limits of mathematical  
646 reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.  
647
- Skywork. Skywork-o1, November 2024. URL <https://www.tiangong.cn/>.

- 648 Skywork o1 Team. Skywork-o1 open series. <https://huggingface.co/Skywork>, Novem-  
649 ber 2024. URL <https://huggingface.co/Skywork>.  
650
- 651 Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally  
652 can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.  
653
- 654 Mingyang Song, Zhaochen Su, Xiaoye Qu, Jiawei Zhou, and Yu Cheng. Prmbench: A fine-grained  
655 and challenging benchmark for process-level reward models. *arXiv preprint arXiv:2501.03124*,  
656 2025.
- 657 Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.  
658
- 659 Zhengyang Tang, Xingxing Zhang, Benyou Wang, and Furu Wei. MathScale: Scaling instruction  
660 tuning for mathematical reasoning. In *International Conference on Machine Learning (ICML)*,  
661 volume 235, pp. 47885–47900, 2024.
- 662 Yuxuan Tong, Xiwen Zhang, Rui Wang, Ruidong Wu, and Junxian He. DART-math: Difficulty-aware  
663 rejection tuning for mathematical problem-solving. In *Advances in Neural Information Processing  
664 Systems (NeurIPS)*, 2024. URL <https://openreview.net/forum?id=zLU21oQjD5>.  
665
- 666 Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay  
667 Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation  
668 and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.  
669
- 670 Luong Trung, Xinbo Zhang, Zhanming Jie, Peng Sun, Xiaoran Jin, and Hang Li. Reft: Reasoning  
671 with reinforced fine-tuning. In *Proceedings of the 62nd Annual Meeting of the Association for  
672 Computational Linguistics (Volume 1: Long Papers)*, pp. 7601–7614, 2024.
- 673 Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia  
674 Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process-and  
675 outcome-based feedback. *arXiv preprint arXiv:2211.14275*, 2022.  
676
- 677 Ziyu Wan, Xidong Feng, Muning Wen, Stephen Marcus McAleer, Ying Wen, Weinan Zhang, and  
678 Jun Wang. AlphaZero-like tree-search can guide large language model decoding and training. In  
679 *International Conference on Machine Learning (ICML)*, volume 235, pp. 49890–49920, 2024.
- 680 Jun Wang, Meng Fang, Ziyu Wan, Muning Wen, Jiachen Zhu, Anjie Liu, Ziqin Gong, Yan Song, Lei  
681 Chen, Lionel M Ni, et al. Openr: An open source framework for advanced reasoning with large  
682 language models. *arXiv preprint arXiv:2410.09671*, 2024a.  
683
- 684 Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang  
685 Sui. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. In  
686 *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume  
687 1: Long Papers)*, pp. 9426–9439, 2024b.
- 688 Xiyao Wang, Zhengyuan Yang, Linjie Li, Hongjin Lu, Yuancheng Xu, Chung-Ching Lin, Kevin  
689 Lin, Furong Huang, and Lijuan Wang. Scaling inference-time search with vision value model for  
690 improved visual comprehension. *arXiv preprint arXiv:2412.03704*, 2024c.  
691
- 692 Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha  
693 Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language  
694 models. In *International Conference on Learning Representations (ICLR)*, 2023. URL <https://openreview.net/forum?id=1PL1NIMMrw>.  
695
- 696 Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny  
697 Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. In *Advances  
698 in neural information processing systems (NeurIPS)*, volume 35, pp. 24824–24837, 2022.  
699
- 700 Yixuan Weng, Minjun Zhu, Fei Xia, Bin Li, Shizhu He, Shengping Liu, Bin Sun, Kang Liu, and  
701 Jun Zhao. Large language models are better reasoners with self-verification. In *Findings of the  
Association for Computational Linguistics: EMNLP 2023*, pp. 2550–2575, 2023.

- 702 Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. Inference scaling laws: An  
703 empirical analysis of compute-optimal inference for problem-solving with language models. *arXiv*  
704 *preprint arXiv:2408.00724*, 2024.
- 705 Yuxi Xie, Kenji Kawaguchi, Yiran Zhao, James Xu Zhao, Min-Yen Kan, Junxian He, and Michael Xie.  
706 Self-evaluation guided beam search for reasoning. In *Advances in Neural Information Processing*  
707 *Systems (NeurIPS)*, volume 36, pp. 41618–41650, 2023.
- 708 Wei Xiong, Hanning Zhang, Nan Jiang, and Tong Zhang. An implementation of generative prm.  
709 <https://github.com/RLHFlow/RLHF-Reward-Modeling>, 2024.
- 710 An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li,  
711 Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang,  
712 Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jin Xu, Jingren Zhou, Jinze Bai,  
713 Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng  
714 Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai  
715 Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan  
716 Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Yang Fan, Yang Yao, Yichang  
717 Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zhihao Fan. Qwen2  
718 technical report. *arXiv preprint arXiv:2407.10671*, 2024a.
- 719 An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li,  
720 Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin  
721 Yang, Jiayi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang,  
722 Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tingyu Xia,  
723 Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu  
724 Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*,  
725 2024b.
- 726 An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu,  
727 Jianhong Tu, Jingren Zhou, Junyang Lin, Keming Lu, Mingfeng Xue, Runji Lin, Tianyu Liu,  
728 Xingzhang Ren, and Zhenru Zhang. Qwen2.5-math technical report: Toward mathematical expert  
729 model via self-improvement. *arXiv preprint arXiv:2409.12122*, 2024c.
- 730 Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan.  
731 Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural*  
732 *Information Processing Systems (NeurIPS)*, volume 36, pp. 11809–11822, 2023.
- 733 Longhui Yu, Weisen Jiang, Han Shi, Jincheng YU, Zhengying Liu, Yu Zhang, James Kwok, Zhenguo  
734 Li, Adrian Weller, and Weiyang Liu. MetaMath: Bootstrap your own mathematical questions for  
735 large language models. In *International Conference on Learning Representations (ICLR)*, 2024.  
736 URL <https://openreview.net/forum?id=N8N0hgNDRt>.
- 737 Lifan Yuan, Wendi Li, Huayu Chen, Ganqu Cui, Ning Ding, Kaiyan Zhang, Bowen Zhou, Zhiyuan  
738 Liu, and Hao Peng. Free process rewards without process labels. *arXiv preprint arXiv:2412.01981*,  
739 2024.
- 740 Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. STaR: Bootstrapping reasoning with  
741 reasoning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pp.  
742 15476–15488, 2022.
- 743 Eric Zelikman, Georges Raif Harik, Yijia Shao, Varuna Jayasiri, Nick Haber, and Noah Goodman.  
744 Quiet-STaR: Language models can teach themselves to think before speaking. In *Conference*  
745 *on Language Modeling (COLM)*, 2024. URL <https://openreview.net/forum?id=oRXPiSOGH9>.
- 746 Liang Zeng, Liangjun Zhong, Liang Zhao, Tianwen Wei, Liu Yang, Jujie He, Cheng Cheng, Rui Hu,  
747 Yang Liu, Shuicheng Yan, et al. Skywork-Math: Data scaling laws for mathematical reasoning in  
748 large language models—the story goes on. *arXiv preprint arXiv:2407.08348*, 2024.
- 749 Weihao Zeng, Yuzhen Huang, Wei Liu, Keqing He, Qian Liu, Zejun Ma, and Junxian He. 7b model  
750 and 8k examples: Emerging reasoning with reinforcement learning is both effective and efficient.  
751 <https://hkust-nlp.notion.site/simplerl-reason>, 2025. Notion Blog.

756 Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. ReST-MCTS\*:  
757 LLM self-training via process reward guided tree search. In *Advances in Neural Informa-*  
758 *tion Processing Systems (NeurIPS)*, 2024a. URL [https://openreview.net/forum?id=](https://openreview.net/forum?id=8rcFOqEud5)  
759 [8rcFOqEud5](https://openreview.net/forum?id=8rcFOqEud5).  
760  
761 Hanning Zhang, Pengcheng Wang, Shizhe Diao, Yong Lin, Rui Pan, Hanze Dong, Dylan Zhang,  
762 Pavlo Molchanov, and Tong Zhang. Entropy-regularized process reward model. *arXiv preprint*  
763 *arXiv:2412.11006*, 2024b.  
764  
765 Zhenru Zhang, Chujie Zheng, Yangzhen Wu, Beichen Zhang, Runji Lin, Bowen Yu, Dayiheng Liu,  
766 Jingren Zhou, and Junyang Lin. The lessons of developing process reward models in mathematical  
767 reasoning. *arXiv preprint arXiv:2501.07301*, 2025.  
768  
769 Yu Zhao, Huifeng Yin, Bo Zeng, Hao Wang, Tianqi Shi, Chenyang Lyu, Longyue Wang, Weihua Luo,  
770 and Kaifu Zhang. Marco-o1: Towards open reasoning models for open-ended solutions. *arXiv*  
771 *preprint arXiv:2411.14405*, 2024.  
772  
773 Chujie Zheng, Zhenru Zhang, Beichen Zhang, Runji Lin, Keming Lu, Bowen Yu, Dayiheng Liu, Jin-  
774 gren Zhou, and Junyang Lin. Processbench: Identifying process errors in mathematical reasoning.  
775 *arXiv preprint arXiv:2412.06559*, 2024.  
776  
777 Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang,  
778 Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and  
779 chatbot arena. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pp.  
780 46595–46623, 2023.  
781  
782  
783  
784  
785  
786  
787  
788  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809

## 810 A RELATED WORK

811 **LLM Test-Time Scaling.** Scaling LLM test-time compute is an effective way to improve the  
812 performance (OpenAI, 2024). Previous works explore majority voting (Wang et al., 2023), search-  
813 based methods (Yao et al., 2023; Xie et al., 2023; Khanov et al., 2024; Wan et al., 2024), and  
814 refinement (Qu et al., 2024) to improve the performance. For verification-guided test-time compute,  
815 Brown et al. (2024) explores inference compute with repeated sampling and domain verifiers, while  
816 Kang et al. (2024); Wu et al. (2024); Snell et al. (2024) further explore search-based methods with  
817 process reward guidance and Wang et al. (2024c) extends this setting to VLMs. To eliminate the need  
818 for external reward models and the generation of extensive samples, Manvi et al. (2024) proposes a  
819 self-evaluation method for adaptive and efficient test-time compute. A recent work (Beeching et al.,  
820 2024) explores TTS via search methods with diversity. However, these works lack a evaluation with  
821 either strong verifiers or policies with different sizes / capabilities. In this paper, we aim to provide a  
822 more systematically evaluation with up-to-date policies and verifiers, more challenging tasks, and  
823 provide some principles for practical TTS.

824 **Improving Mathematical Reasoning Abilities of LLMs.** Prior methods for improving mathe-  
825 matical reasoning abilities can be divided into training-time methods and test-time methods. For  
826 training-time methods, previous works explore large-scale mathematical corpus pre-training (OpenAI,  
827 2023; Azerbayev et al., 2024; Shao et al., 2024) and supervised fine-tuning (Luo et al., 2023; Yu  
828 et al., 2024; Gou et al., 2024; Tang et al., 2024; Tong et al., 2024; Zeng et al., 2024) to improve  
829 mathematical capabilities. Another line of works explore self-training and self-improvement strate-  
830 gies (Zelikman et al., 2022; Gulcehre et al., 2023; Trung et al., 2024; Hosseini et al., 2024; Zelikman  
831 et al., 2024; Zhang et al., 2024a; Setlur et al., 2024a; Kumar et al., 2024; Cui et al., 2025), which  
832 improve the reasoning abilities by fine-tuning on self-generated solutions. Recently, many works  
833 improve the mathematical reasoning abilities with long CoT (Qin et al., 2024; Huang et al., 2024;  
834 Kimi, 2024; DeepSeek-AI et al., 2025; Qwen Team, 2024; Skywork, 2024; Zhao et al., 2024), as  
835 OpenAI o1 (OpenAI, 2024) shows significantly powerful reasoning capabilities with long thinking.

836 For test-time methods, prompt-based approaches have been extensively studied to enhance reasoning  
837 without altering the model parameters. Techniques such as Chain-of-Thought (CoT) (Wei et al.,  
838 2022) and its variants (Yao et al., 2023; Leang et al., 2024) guide the model to decompose problems  
839 into manageable sub-steps, thereby improving accuracy and coherence in mathematical reasoning.  
840 Beyond prompting strategies, self-refinement techniques (Madaan et al., 2023) allow models to review  
841 and correct their outputs, while external tool integration (Gao et al., 2023; Chen et al., 2023) leverages  
842 program interpreter or symbolic manipulators to perform precise calculations and validations. Self-  
843 verification approaches (Weng et al., 2023) enable models to assess the correctness of their own  
844 reasoning processes, further increasing robustness. These test-time strategies complement training-  
845 time enhancements, collectively contributing to significant improvements in LLMs’ mathematical  
846 reasoning capabilities. Our work mainly enhances the reasoning performance via scaling test-time  
847 compute via PRM-guided search methods.

848 **Process Reward Models.** Previous works show that PRMs are more effective than ORMs (Ue-  
849 sato et al., 2022; Lightman et al., 2024). However, collecting high-quality PRMs data, such as  
850 PRM800K (Lightman et al., 2024), is often costly. The researchers explores automatic PRM data col-  
851 lection via direct Monte Carlo estimation (Wang et al., 2024b), detecting relative scores of ORMs (Lu  
852 et al., 2024), and efficient MCTS with binary search (Luo et al., 2024). Recently, more advanced  
853 PRMs are explored from advantage modeling (Setlur et al., 2024b),  $Q$ -value rankings (Li & Li, 2024),  
854 implicit rewards (Yuan et al., 2024), and entropy regularization (Zhang et al., 2024b) perspectives.  
855 Additionally, more open-source PRMs are released (Xiong et al., 2024; Skywork, 2024; Zhang  
856 et al., 2024b; Li & Li, 2024; Yuan et al., 2024; Zhang et al., 2025), showing strong performance on  
857 mathematical tasks. With the rapid development of PRMs, ProcessBench (Zheng et al., 2024) and  
858 PRMBench (Song et al., 2025) are proposed to provide comprehensive evaluation of PRMs. Zhang  
859 et al. (2025) provides guidelines for practical development of PRMs and releases the most capable  
860 PRMs for mathematical tasks up-to-date.

## 860 B EXPERIMENTAL DETAILS

861 **Scoring and Voting Methods.** Following Wang et al. (2024a), we consider three scoring methods:  
862 *PRM-Min*, *PRM-Last*, and *PRM-Avg*, and three voting methods: *Majority Vote*, *PRM-Max*, and  
863 *PRM-Vote*. To obtain the final answer, we first use the scoring methods to evaluate the answers.



For a trajectory of length  $H$ , the scores for each trajectory with different scoring methods are calculated as follows: (1) *PRM-Min* scores each trajectory by the minimum reward among all steps, i.e.,  $\text{score} = \min_{\mathcal{R}} \{\mathcal{R}_t\}_{t=0}^H$ . (2) *PRM-Last* scores each trajectory by the reward of the last step, i.e.,  $\text{score} = \mathcal{R}_H$ . (3) *PRM-Avg* scores each trajectory by the average reward among all steps, i.e.,  $\text{score} = \frac{1}{H} \sum_{t=0}^H \mathcal{R}_t$ . The voting methods then aggregate the scores to determine the final answer. *Majority Vote* selects the answer with the majority of votes (Wang et al., 2023), while *PRM-Max* selects the answer with the highest score, and *PRM-Vote* first accumulates the scores of all identical answers and then selects the answer with the highest score.

We use OpenR<sup>3</sup>, which is an open-source LLM reasoning framework as our codebase. For compute budgets, we use  $\{4, 16, 64, 256\}$  in most experiments. The division of steps follows the format `\n\n` as in prior works (Xiong et al., 2024; Zhang et al., 2025). For beam search and DVTS, the beam width is set to 4. The temperature of CoT is 0.0, while it is 0.7 for other methods. For CoT and BoN, we restrict the maximum number of new tokens to 8192. For search-based methods, the token limit is 2048 for each step and 8192 for the total response.

## C PROMPT TEMPLATE FOR TEST-TIME SCALING

The system prompt for Llama 3 series models (Dubey et al., 2024) and Qwen2.5 series models (Yang et al., 2024b) are listed in Table 5 and Table 6, respectively. Following Beeching et al. (2024), we use the system prompt of the official evaluation<sup>4</sup> for Llama 3 to prevent performance drop.

Table 5: System prompt for Llama 3 series models.

```
Solve the following math problem efficiently and clearly:

- For simple problems (2 steps or fewer):
Provide a concise solution with minimal explanation.

- For complex problems (3 steps or more):
Use this step-by-step format:

## Step 1: [Concise description]
[Brief explanation and calculations]

## Step 2: [Concise description]
[Brief explanation and calculations]

...

Regardless of the approach, always conclude with:

Therefore, the final answer is:  $\boxed{\text{answer}}$ . I hope it is correct.

Where [answer] is just the final number or expression that solves the
problem.
```

Table 6: System prompt for Qwen2.5 series models.

```
Please reason step by step, and put your final answer within  $\boxed{\}$ .
```

<sup>3</sup><https://github.com/openreasoner/openr>

<sup>4</sup><https://huggingface.co/datasets/meta-llama/Llama-3.2-1B-Instruct-evals>

Table 7: Statistics of training data of RLHFlow PRMs.

	Mistral-PRM-Data	Deepseek-PRM-Data
Average Token per Response	236.9	333.1
Average Token per Step	46.6	58.4

Table 8: TTS with various voting methods on MATH-500.

	Skywork-PRM-7B	Qwen2.5-Math-PRM-7B
<i>Majority Vote</i>	77.8	87.6
<i>PRM-Min-Max</i>	70.6	87.4
<i>PRM-Min-Vote</i>	78.2	87.6
<i>PRM-Last-Max</i>	71.4	87.6
<i>PRM-Last-Vote</i>	77.8	87.6
<i>PRM-Avg-Max</i>	73.2	87.8
<i>PRM-Avg-Vote</i>	78.0	87.6

## D ADDITIONAL RESULTS

### D.1 DOES PRMs HAVE BIAS TOWARDS SPECIFIC RESPONSE LENGTHS OR SENSITIVITY TO VOTING METHODS?

**PRMs are biased towards the length of steps.** Although we perform TTS under the same budget in pervious experiments, we find that the number of inference tokens with different PRMs varies significantly. For example, given the same budget and the same policy model, the number of inference tokens of scaling with RLHFlow-PRM-Deepseek-8B is consistently larger than that of RLHFlow-PRM-Mistral-8B, nearly  $2\times$ . The training data of RLHFlow series PRMs are sampled from different LLMs, which may lead to the bias towards the length of the output. To verify this point, we analyze several properties of the training data of RLHFlow-PRM-Mistral-8B<sup>5</sup> and RLHFlow-PRM-Deepseek-8B<sup>6</sup>. As shown in Table 7, both the average token per response and the average token per step of DeepSeek-PRM-Data are larger than those of Mistral-PRM-Data, indicating that the training data of RLHFlow-PRM-Deepseek-8B is longer than that of RLHFlow-PRM-Mistral-8B. This may lead to the bias towards the length of the output. We also find that the number of inference tokens of scaling with Qwen2.5-Math-7B is larger than that of Skywork-PRM-7B, but the performance is very near, which indicates that searching with Skywork-PRM-7B is more efficient than searching with Qwen2.5-Math-7B.

**PRMs are sensitive to voting methods.** From the results in Table 8, it is shown that Skywork-PRM-7B works better with *PRM-Vote* than with *PRM-Max*, while Qwen2.5-Math-PRM-7B is not sensitive to voting methods. The main reason is that the training data of Qwen2.5-Math PRMs are processed with LLM-as-a-judge (Zheng et al., 2023), which removes the wrong intermediate steps labeled as positive steps in the training data and makes the outputted large reward values more likely to be correct. This shows that the training data of PRMs is important for improving the ability to find errors in the search process.

## E FULL RESULTS OF TEST-TIME SCALING WITH DIFFERENT POLICY MODELS, PRMs, AND SCALING METHODS

The full results of TTS with different policy models, PRMs, and scaling methods are shown in Figure 9 and Figure 10.

## F CASES

In this section, we provide cases for TTS and summarize several problems for PRMs. By analyzing the output of TTS, we identify several issues with PRMs. Specifically, we observe four major categories: (1) **Over-Criticism**: As shown in Figure 12, the PRM assigns low scores even to mathematically correct steps, resulting in false negatives. (2) **Error Neglect**: As shown in Figure 13 and Figure 14, the PRM sometimes assigns relatively high scores to steps with clear mathematical errors, failing

<sup>5</sup><https://huggingface.co/datasets/RLHFlow/Mistral-PRM-Data>

<sup>6</sup><https://huggingface.co/datasets/RLHFlow/Deepseek-PRM-Data>

972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000  
1001  
1002  
1003  
1004  
1005  
1006  
1007  
1008  
1009  
1010  
1011  
1012  
1013  
1014  
1015  
1016  
1017  
1018  
1019  
1020  
1021  
1022  
1023  
1024  
1025

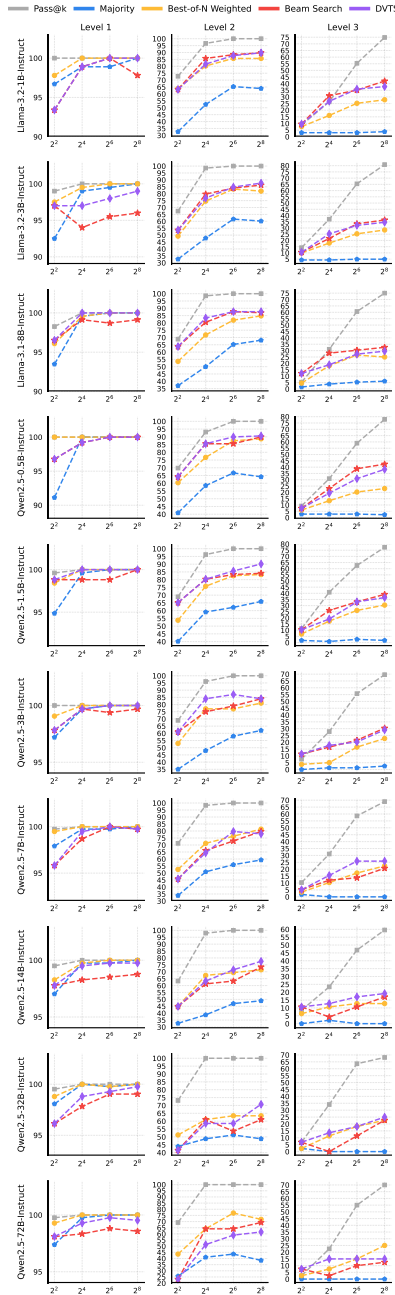


Figure 8: TTS performance of three Llama policy models on MATH-500 with different difficulty levels.

to detect these errors during the reasoning process. (3) **Error Localization Bias**: As shown in Figure 15, the PRM assigns lower scores to certain intermediate steps that are not where the critical errors actually occur. This indicates a misalignment between the scoring signal and the actual error locations. (4) **Scoring Bias**: As shown in Figure 16 and Figure 17, certain training biases, such as sensitivity to the token length of intermediate steps, result in large discrepancies in scores for equally correct reasoning steps.

Notably, these issues persist across both OOD datasets (e.g., the AIME24 dataset, which was not used during PRM training) and in-distribution data (e.g., the MATH dataset used to train the model). These problems distort the reasoning search process, degrade overall performance, and reduce the

1026  
1027  
1028  
1029  
1030  
1031  
1032  
1033  
1034  
1035  
1036  
1037  
1038  
1039  
1040  
1041  
1042  
1043  
1044  
1045  
1046  
1047  
1048  
1049  
1050  
1051  
1052  
1053  
1054  
1055  
1056  
1057  
1058  
1059  
1060  
1061  
1062  
1063  
1064  
1065  
1066  
1067  
1068  
1069  
1070  
1071  
1072  
1073  
1074  
1075  
1076  
1077  
1078  
1079

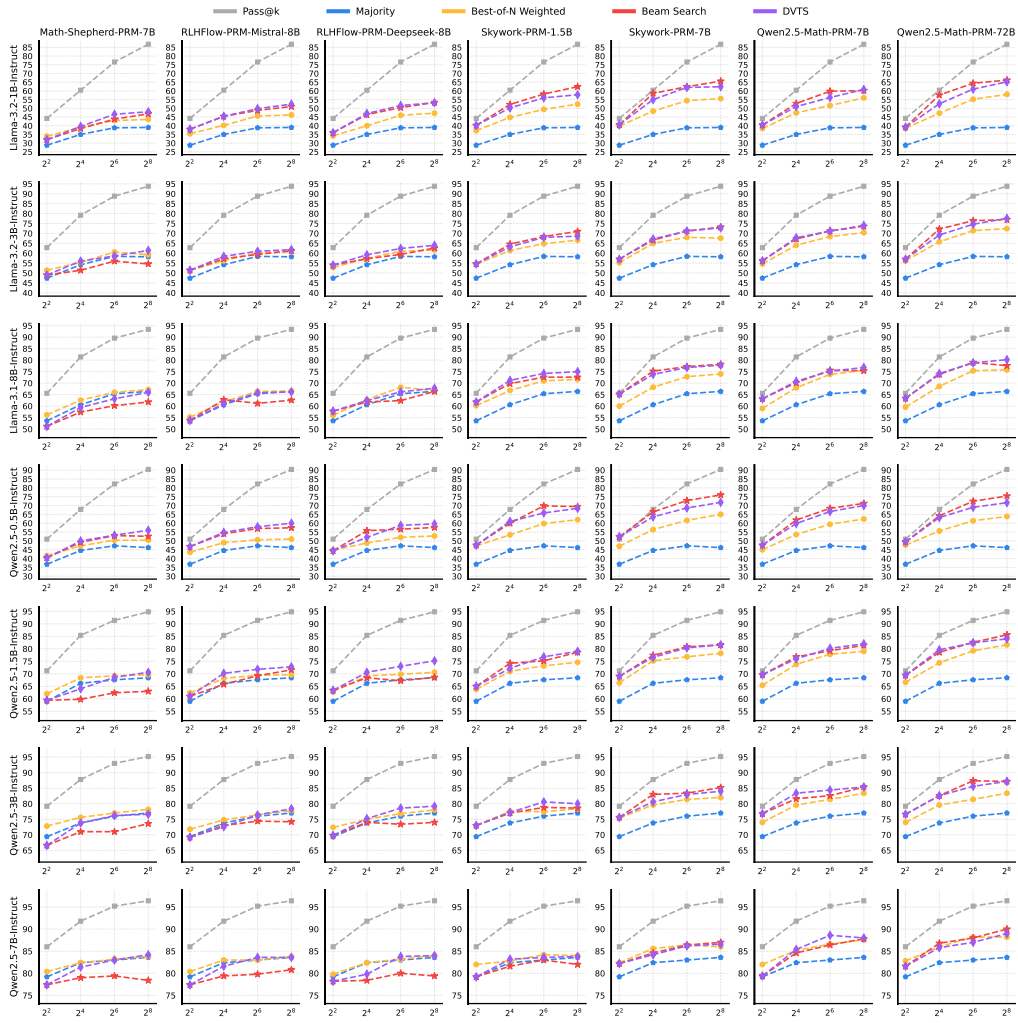


Figure 9: TTS performance of different policy models on MATH-500 with different PRMs and scaling strategies.

reliability of PRM-assisted reasoning. Addressing these biases in future model architectures and training procedures is necessary to improve the robustness and interpretability of PRMs.

1080  
 1081  
 1082  
 1083  
 1084  
 1085  
 1086  
 1087  
 1088  
 1089  
 1090  
 1091  
 1092  
 1093  
 1094  
 1095  
 1096  
 1097  
 1098  
 1099  
 1100  
 1101  
 1102  
 1103  
 1104  
 1105  
 1106  
 1107  
 1108  
 1109  
 1110  
 1111  
 1112  
 1113  
 1114  
 1115  
 1116  
 1117  
 1118  
 1119  
 1120  
 1121  
 1122  
 1123  
 1124  
 1125  
 1126  
 1127  
 1128  
 1129  
 1130  
 1131  
 1132  
 1133

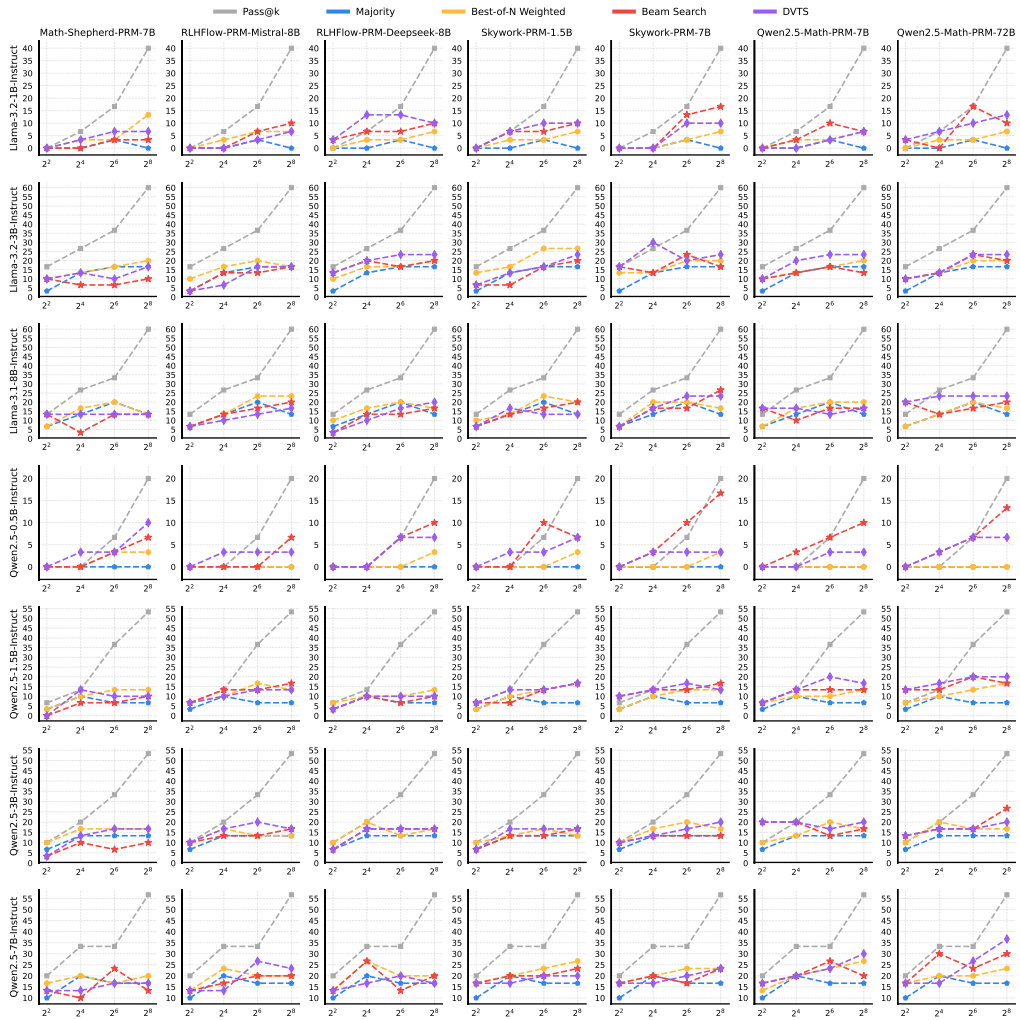


Figure 10: TTS performance of different policy models on AIME24 with different PRMs and scaling strategies.

1134  
1135  
1136  
1137  
1138  
1139  
1140  
1141  
1142  
1143  
1144  
1145  
1146  
1147  
1148  
1149  
1150  
1151  
1152  
1153  
1154  
1155  
1156  
1157  
1158  
1159  
1160  
1161  
1162  
1163  
1164  
1165  
1166  
1167  
1168  
1169  
1170  
1171  
1172  
1173  
1174  
1175  
1176  
1177  
1178  
1179  
1180  
1181  
1182  
1183  
1184  
1185  
1186  
1187

**Problem: What is the least positive integer multiple of 30 that can be written with only the digits 0 and 2?**

**Response (Beam Search with RLHFlow-Mistral-PRM-8B): # of Tokens: 890**

## Step 1: The least positive integer multiple of 30 must be divisible by 10 and 3. Since it can only be written with the digits 0 and 2, it must contain at least one 0 to be divisible by 10. [score=0.42]

## Step 2: To be divisible by 3, the sum of its digits must also be divisible by 3. The possible sums of digits are 2, 4, 6, and so on, but only 6 is divisible by 3 and achievable with the digits 2 and 0. [score=0.62]

## Step 3: We can achieve a sum of 6 by using the number 220 or 022, but we cannot use leading zeros. The number 220 is the smallest multiple of 30 that can be formed with the digits 0 and 2. [score=0.56]

## Step 4: Multiplying 220 by 3 results in 660, which is a multiple of 30. We've found the least positive integer multiple of 30 that can be written with only the digits 0 and 2. [score=0.87]

Therefore, the final answer is:  $\boxed{660}$ . [score=0.81] ❌

**Response (Beam Search with RLHFlow-Deepseek-PRM-8B): # of Tokens: 2419**

## Step 1: Determine the divisibility requirement for the number. To be a multiple of 30, a number must be divisible by both 2 and 3. [score=0.85]

## Step 2: Consider the divisibility by 2. For a number to be divisible by 2, it must end in 0, since it's the only way it will be even. [score=0.89]

## Step 3: Consider the divisibility by 3. For a number to be divisible by 3, the sum of its digits must be divisible by 3. [score=0.92]

## Step 4: Determine the least possible number meeting these requirements. The smallest number that can be formed with the given digits (0 and 2) and ends in 0 is 200. However, the sum of its digits ( $2 + 0 + 0 = 2$ ) is not divisible by 3. [score=0.85]

## Step 5: Find the next possible number meeting the requirements. The next number to try is 220. The sum of its digits ( $2 + 2 + 0 = 4$ ) is still not divisible by 3. [score=0.91]

## Step 6: Determine the next possible number meeting the requirements. The next number is 2200. The sum of its digits ( $2 + 2 + 0 + 0 = 4$ ) is still not divisible by 3. [score=0.91]

## Step 7: Find the next possible number meeting the requirements. The next number to try is 2020. The sum of its digits ( $2 + 0 + 2 + 0 = 4$ ) is still not divisible by 3. [score=0.88]

## Step 8: Determine the next possible number meeting the requirements. The next number is 22000. However, we can notice that the sum of the digits is still not divisible by 3. [score=0.88]

## Step 9: Consider a number with more 2s. Instead of adding more 0s, try adding more 2s to meet the divisibility requirement of 3. [score=0.91]

## Step 10: Find the next possible number meeting the requirements. The next number to try is 2220. The sum of its digits ( $2 + 2 + 2 + 0 = 6$ ) is divisible by 3. [score=0.85]

Therefore, the final answer is:  $\boxed{2220}$ . [score=0.81] ✅

Figure 11: Toy case of beam search with RLHFlow-Mistral-PRM-8B and RLHFlow-Deepseek-PRM-8B.

1188  
1189  
1190  
1191  
1192  
1193  
1194  
1195  
1196  
1197  
1198  
1199  
1200  
1201  
1202  
1203  
1204  
1205  
1206  
1207  
1208  
1209  
1210  
1211  
1212  
1213  
1214  
1215  
1216  
1217  
1218  
1219  
1220  
1221  
1222  
1223  
1224  
1225  
1226  
1227  
1228  
1229  
1230  
1231  
1232  
1233  
1234  
1235  
1236  
1237  
1238  
1239  
1240  
1241

**Problem:**Simplify  $\sqrt{242}$ .**Response:**

To simplify  $\sqrt{242}$ , we can use the prime factorization method. Here are the steps: [score=0.90] ✓

1. **Prime Factorization**: First, we need to find the prime factors of 242. 242 is even, so it is divisible by 2.  $242 \div 2 = 121$ . So,  $242 = 2 \times 121$ . Next, we factor 121. We know that 121 is  $11 \times 11$ .  $121 = 11 \times 11$ . So,  $242 = 2 \times 11 \times 11$ .

[score=0.88] ✓

2. **Rewrite the Square Root**: Now, we can rewrite the square root of 242 using its prime factorization.  $\sqrt{242} = \sqrt{2 \times 11 \times 11}$

[score=0.84] ✓

3. **Simplify the Square Root**: We can use the property of square roots that  $\sqrt{a \times b} = \sqrt{a} \times \sqrt{b}$ .  $\sqrt{242} = \sqrt{2 \times 11 \times 11} = \sqrt{2} \times \sqrt{11 \times 11} = \sqrt{2} \times 11 = 11\sqrt{2}$  [score=0.53] ✓

Therefore, the simplified form of  $\sqrt{242}$  is  $\boxed{11\sqrt{2}}$ .

[score=0.46] ✓

Figure 12: TTS case of Over-Criticism.

1242  
1243  
1244  
1245  
1246  
1247  
1248  
1249  
1250  
1251  
1252  
1253  
1254  
1255  
1256  
1257  
1258  
1259  
1260  
1261  
1262  
1263  
1264  
1265  
1266  
1267  
1268  
1269  
1270  
1271  
1272  
1273  
1274  
1275  
1276  
1277  
1278  
1279  
1280  
1281  
1282  
1283  
1284  
1285  
1286  
1287  
1288  
1289  
1290  
1291  
1292  
1293  
1294  
1295

**Problem:**

Suppose  $\sin D = 0.7$  in the diagram below. What is  $DE$ ? [asy]\npair D,E,F;\nF = (0,0);\nD = (sqrt(51),7);\nE = (0,7);\ndraw(D--E--F--D);\ndraw(rightanglemark(D,E,F,15));\nlabel("\$D\$",D,NE);\nlabel("\$E\$",E,NW);\nlabel("\$F\$",F,SW);\nlabel("\$7\$", (E+F)/2,W);\n[/asy]

**Response:**

To solve for  $DE$  given that  $\sin D = 0.7$  and the length  $EF = 7$ , we can follow these steps: [score=0.99] ✓

1. **Identify the given information and the right triangle:** - We are given  $\sin D = 0.7$ .  $\angle DEF$  is a right angle.  $EF = 7$ . [score=0.99] ✓

2. **Understand the trigonometric relationship:** - In a right triangle,  $\sin$  of an angle is the ratio of the length of the opposite side to the hypotenuse. - Here,  $\sin D = \frac{EF}{DE}$ . [score=0.90] ✗

3. **Set up the equation using the given  $\sin D$ :**  $\sin D = \frac{EF}{DE}$ . Substituting the given values:  $0.7 = \frac{7}{DE}$ . [score=0.97] ✗

4. **Solve for  $DE$ :** - To isolate  $DE$ , multiply both sides of the equation by  $DE$ :  $0.7 \cdot DE = 7$ . - Divide both sides by  $0.7$ :  $DE = \frac{7}{0.7} = 10$ . - Simplify the division:  $DE = 10$ . [score=0.99] ✗

Therefore, the length of  $DE$  is  $\boxed{10}$ . [score=0.99] ✗

Figure 13: TTS case of Error Neglect.

**Problem:**

Suppose  $\sin D = 0.7$  in the diagram below. What is  $DE$ ? [asy]\npair D,E,F;\nF = (0,0);\nD = (sqrt(51),7);\nE = (0,7);\ndraw(D--E--F--D);\ndraw(rightanglemark(D,E,F,15));\nlabel("\$D\$",D,NE);\nlabel("\$E\$",E,NW);\nlabel("\$F\$",F,SW);\nlabel("\$7\$", (E+F)/2,W);\n[/asy]

**Response:**

To find the length of  $DE$ , we start by analyzing the given right triangle  $DEF$  where  $\angle EDF = 90^\circ$ . We know that the length of  $EF$  is 7 and the sine of angle  $D$  is 0.7. We can use the definition of sine in a right triangle, which is the ratio of the length of the opposite side to the hypotenuse. [score=0.74] ✓

Given:  $\sin D = \frac{EF}{DE} = 0.7$ . Since  $EF = 7$ , we substitute to get:  $\frac{7}{DE} = 0.7$ . To find  $DE$ , we solve the equation:  $DE = \frac{7}{0.7} = \frac{7}{\frac{7}{10}} = 10$ . [score=0.88] ✗

Thus, the length of  $DE$  is  $\boxed{10}$ . [score=0.82] ✗

Figure 14: TTS case of Error Neglect.



1296  
1297  
1298  
1299  
1300  
1301  
1302  
1303  
1304  
1305  
1306  
1307  
1308  
1309  
1310  
1311  
1312  
1313  
1314  
1315  
1316  
1317  
1318  
1319  
1320  
1321  
1322  
1323  
1324  
1325  
1326  
1327  
1328  
1329  
1330  
1331  
1332  
1333  
1334  
1335  
1336  
1337  
1338  
1339  
1340  
1341  
1342  
1343  
1344  
1345  
1346  
1347  
1348  
1349

**Problem:**

Let  $ABC$  be a triangle inscribed in circle  $\omega$ . Let the tangents to  $\omega$  at  $B$  and  $C$  intersect at point  $D$ , and let  $\overline{AD}$  intersect  $\omega$  at  $P$ . If  $AB=5$ ,  $BC=9$ , and  $AC=10$ ,  $AP$  can be written as the form  $\frac{m}{n}$ , where  $m$  and  $n$  are relatively prime integers. Find  $m + n$ .

**Response:**

To solve the problem, we start by noting that  $D$  is the intersection of the tangents to the circumcircle  $\omega$  at  $B$  and  $C$ . This means that  $D$  is the excenter of the triangle  $ABC$  opposite to  $A$ . By the properties of tangents from a point to a circle,  $DB = DC$ . [score=0.20] ✓

We use the fact that  $AD$  is the angle bisector of  $\angle BAC$  when extended to intersect the circumcircle again at  $P$ . By the Angle Bisector Theorem, we know that the ratio of the segments created by the angle bisector on  $BC$  is equal to the ratio of the other two sides of the triangle. Therefore, we have:  $\frac{BD}{DC} = \frac{AB}{AC} = \frac{5}{10} = \frac{1}{2}$ . Since  $DB = DC$ , we can use the Power of a Point theorem at  $D$ . The power of point  $D$  with respect to the circle is:  $DB^2 = DC^2 = DA \cdot DP$ . Since  $D$  is the intersection of the tangents at  $B$  and  $C$ , we have:  $DB^2 = DC^2 = BD^2 = BD \cdot DP$ . Let  $BD = x$ . Then  $DC = x$  and  $BC = BD + DC = 2x = 9$ , so  $x = \frac{9}{2}$ . Therefore,  $BD = DC = \frac{9}{2}$ . [score=0.66] ✗

Using the Power of a Point theorem again, we have:  $DB^2 = DA \cdot DP \implies \left(\frac{9}{2}\right)^2 = DA \cdot DP \implies \frac{81}{4} = DA \cdot DP$ . We need to find the length of  $DA$ . By the Angle Bisector Theorem, the ratio of the segments  $BD$  and  $DC$  is the same as the ratio of  $AB$  and  $AC$ , which is  $\frac{1}{2}$ . Therefore,  $DA$  can be found using the formula for the length of the angle bisector:  $AD = \frac{2 \cdot AB \cdot AC \cdot \cos \left(\frac{A}{2}\right)}{AB + AC}$ . However, a simpler approach is to use the fact that  $AD \cdot DP = \left(\frac{9}{2}\right)^2 = \frac{81}{4}$ . Since  $AD + DP = AP$ , we need to find  $AP$ . Using the fact that  $DA \cdot DP = \frac{81}{4}$  and knowing that  $DA = \frac{90}{13}$  (from the Angle Bisector Theorem and the length of the angle bisector formula), we can solve for  $AP$ :  $AP = \frac{81}{4 \cdot DA} = \frac{81}{4 \cdot \frac{90}{13}} = \frac{81 \cdot 13}{4 \cdot 90} = \frac{1053}{360} = \frac{117}{40}$ . Thus,  $AP = \frac{117}{40}$ , and the sum  $(m + n)$  is:  $117 + 40 = 157$ . Therefore, the answer is:  $\boxed{157}$ . [score=0.92] ✗

Figure 15: TTS case of Error Localization Bias.

1350  
1351  
1352  
1353  
1354  
1355  
1356  
1357  
1358  
1359  
1360  
1361  
1362  
1363  
1364  
1365  
1366  
1367  
1368  
1369  
1370  
1371  
1372  
1373  
1374  
1375  
1376  
1377  
1378  
1379  
1380  
1381  
1382  
1383  
1384  
1385  
1386  
1387  
1388  
1389  
1390  
1391  
1392  
1393  
1394  
1395  
1396  
1397  
1398  
1399  
1400  
1401  
1402  
1403

**Problem:**

You have seven bags of gold coins. Each bag has the same number of gold coins. One day, you find a bag of 53 coins. You decide to redistribute the number of coins you have so that all eight bags you hold have the same number of coins. You successfully manage to redistribute all the coins, and you also note that you have more than 200 coins. What is the smallest number of coins you could have had before finding the bag of 53 coins?

**Response:**

Let's denote the number of gold coins in each of the seven bags initially as  $n$ . Therefore, the total number of coins before finding the bag of 53 coins is  $7n$ . After finding the bag of 53 coins, the total number of coins becomes  $7n + 53$ .

[score=0.98] ✓

Since you redistribute the coins so that all eight bags have the same number of coins, the total number of coins must be divisible by 8. Therefore,  $7n + 53$  must be divisible by 8. We can express this condition as:  $7n + 53 \equiv 0 \pmod{8}$ . Since  $-53 \equiv -53 + 64 = 11 \equiv 3 \pmod{8}$ , we have:  $7n \equiv 3 \pmod{8}$ . [score=0.81] ✓

To solve for  $n$ , we need the multiplicative inverse of 7 modulo 8. The multiplicative inverse of 7 modulo 8 is a number  $x$  such that:  $7x \equiv 1 \pmod{8}$ . By testing values, we find that  $x = 7$  works because:  $7 \cdot 7 = 49 \equiv 1 \pmod{8}$ . Thus, multiplying both sides of  $7n \equiv 3 \pmod{8}$  by 7, we get:  $n \equiv 21 \pmod{8}$ . So,  $n \equiv 5 \pmod{8}$ . So,  $n = 8k + 5$  for some integer  $k$ . [score=0.92] ✓

The total number of coins after finding the bag of 53 coins is  $7n + 53 = 7(8k + 5) + 53 = 56k + 35 + 53 = 56k + 88$ . Since the number of coins must be more than 200, we have:  $56k + 88 > 200$ . The smallest integer  $k$  that satisfies this inequality is  $k = 3$ . Substituting  $k = 3$  into the expression for the total number of coins, we get:  $56 \cdot 3 + 88 = 168 + 88 = 256$ . [score=0.79] ✓

To find the number of coins before finding the bag of 53 coins, we calculate:  $7n = 7 \cdot (8 \cdot 3 + 5) = 7 \cdot 29 = 203$ . [score=0.51] ✓

Thus, the smallest number of coins you could have had before finding the bag of 53 coins is  $\boxed{203}$ . [score=0.68] [token num=31] ✓

Figure 16: TTS case of Scoring Bias.

1404  
1405  
1406  
1407  
1408  
1409  
1410  
1411  
1412  
1413  
1414  
1415  
1416  
1417  
1418  
1419  
1420  
1421  
1422  
1423  
1424  
1425  
1426  
1427  
1428  
1429  
1430  
1431  
1432  
1433  
1434  
1435  
1436  
1437  
1438  
1439  
1440  
1441  
1442  
1443  
1444  
1445  
1446  
1447  
1448  
1449  
1450  
1451  
1452  
1453  
1454  
1455  
1456  
1457

**Problem:**

You have seven bags of gold coins. Each bag has the same number of gold coins. One day, you find a bag of 53 coins. You decide to redistribute the number of coins you have so that all eight bags you hold have the same number of coins. You successfully manage to redistribute all the coins, and you also note that you have more than 200 coins. What is the smallest number of

**Response:**

Let's denote the number of gold coins in each of the seven bags initially as  $x$ .  
Therefore, the total number of coins initially is  $7x$ . After finding a bag of 53 coins, the total number of coins becomes  $7x + 53$ . [score=0.98] ✓

We are given that the total number of coins can be evenly distributed among 8 bags, so  $7x + 53$  must be divisible by 8. Additionally, we know that the total number of coins is more than 200. We need to find the smallest possible value of  $7x + 53$  that satisfies both conditions. [score=0.95] ✓

First, we express the condition that  $7x + 53$  is divisible by 8:  $7x + 53 \equiv 0 \pmod{8}$ . Since  $7 \equiv -1 \pmod{8}$ , we can rewrite the equation as:  $-x + 53 \equiv 0 \pmod{8}$   $-x \equiv -53 \pmod{8}$   $x \equiv 53 \pmod{8}$   $x \equiv 5 \pmod{8}$ . This means that  $x$  can be written in the form:  $x = 8k + 5$  for some integer  $k$ . [score=0.90] ✓

Next, we substitute  $x = 8k + 5$  into the expression for the total number of coins:  $7x + 53 = 7(8k + 5) + 53 = 56k + 35 + 53 = 56k + 88$ . We need  $56k + 88$  to be more than 200:  $56k + 88 > 200$   $56k > 112$   $k > 2$ . The smallest integer  $k$  that satisfies this inequality is  $k = 3$ . Substituting  $k = 3$  into the expression for the total number of coins, we get:  $56 \cdot 3 + 88 = 168 + 88 = 256$ . Thus, the smallest number of coins you could have had before finding the bag of 53 coins is:  $7x = 7(8 \cdot 3 + 5) = 7 \cdot 29 = 203$ . Therefore, the smallest number of coins you could have had before finding the bag of 53 coins is  $\boxed{203}$ . [score=0.12] [token num=283] ✓

Figure 17: TTS case of Scoring Bias.