Rethinking Compute-Optimal Test-Time Scaling for Mathematical Reasoning

Anonymous ACL submission

Abstract

Test-Time Scaling (TTS) is an important method for improving the performance of Large Language Models (LLMs) by using additional computation during inference. 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 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**. 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.

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).

TTS approaches can be divided into two main categories: (1) Internal TTS, which trains the LLMs to "think" slowly with long Chain-of-Thought (CoT) (OpenAI, 2024; DeepSeek-AI et al., 2025), and (2) External TTS, which improves the reasoning performance via sampling or search-based methods with fixed LLMs (Wu et al., 2025; Snell et al., 2025). The key challenge of external TTS is how to scale compute optimally, that is, allocating the optimal computation for each problem (Snell et al., 2025). 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., 2025; Snell et al., 2025; Beeching et al., 2024). These TTS methods involve several important factors, such as policy models[1], 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

[1]Following Snell et al. (2025), we use "policy models" to refer to LLMs that generate solutions, and "verifiers" for PRMs.

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 and 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, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the transition function, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, and $\gamma \in [0, 1]$ is the discount factor. Given a prompt $x \sim \mathcal{X}$, the policy with parameters θ generates the initial action $a_1 \sim \pi_\theta(\cdot \mid s_1)$, 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 $\tau = \{a_1, a_2, \cdots, a_H\}$.

2.2 Test-Time Scaling Method

We consider three TTS methods: Best-of-N (BoN) (Brown et al., 2024), beam search (Snell et al., 2025), and Diverse Verifier Tree Search (DVTS) (Beeching et al., 2024) and the details of these methods are shown in Appendix A. As pointed out by Snell et al. (2025), 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).

2.3 Compute-Optimal Test-Time Scaling

To maximize the performance of TTS, Snell et al. (2025) propose 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 $\mathrm{Target}(\theta, N, x)$ represent the output distribution over x produced by the policy model with parameters θ and a compute budget of N.

$$\theta^*_{x,y^*(x)}(N) = \arg\max_{\theta} \left(\mathbb{E}_{y\sim\mathrm{Target}(\theta,N,x)} \left[\mathbb{1}_{y=y^*(x)} \right] \right), \tag{1}$$

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., 2025). Previous works on TTS use a single PRM as verifier (Snell et al., 2025; Wu et al., 2025; Beeching et al., 2024). Snell et al. (2025) 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. (2025); Beeching et al. (2024) use a PRM trained on a different policy model to do TTS. From the perspective of Reinforcement Learning (RL), we obtain an *online* PRM in the former case and an *offline* PRM in the latter case. The online PRM predicts more accurate rewards for the responses of the policy model, while the rewards given by an offline PRM are often inaccurate due to out-of-distribution (OOD) issues (Snell et al., 2025; Zheng et al., 2024).

For the practical usage 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 final responses and accuracy; (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:

$$\theta^*_{x,y^*(x),\mathcal{R}}(N) = \arg\max_{\theta} \left(\mathbb{E}_{y\sim\text{Target}(\theta,N,x,\mathcal{R})}\left[\mathbb{1}_{y=y^*(x)}\right]\right), \tag{2}$$

where $\text{Target}(\theta, N, x, \mathcal{R})$ represents the output distribution of the policy model θ, 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. (2025) group problems into five difficulty levels based on Pass@1 accuracy quantiles. However, we find that using difficulty levels

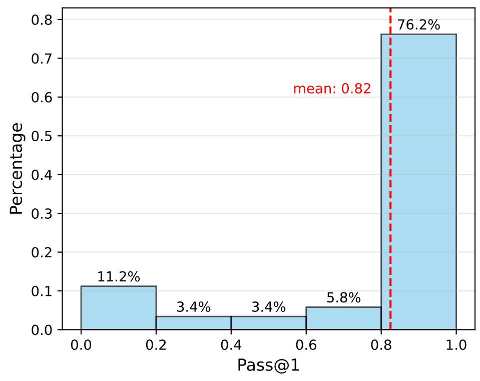

Figure 1: Distribution of Pass@1 accuracy of Qwen2.5-72B-Instruct on MATH-500, divided into five bins.

from MATH (Hendrycks et al., 2021) or oracle labels based on Pass@1 accuracy quantiles (Snell et al., 2025) is not effective since different policy models have different reasoning capabilities. As shown in Figure 1, 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% $\sim$ 100%), medium (10% $\sim$ 50%), and hard (0% $\sim$ 10%).

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. (2025); Beeching et al. (2024).

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.

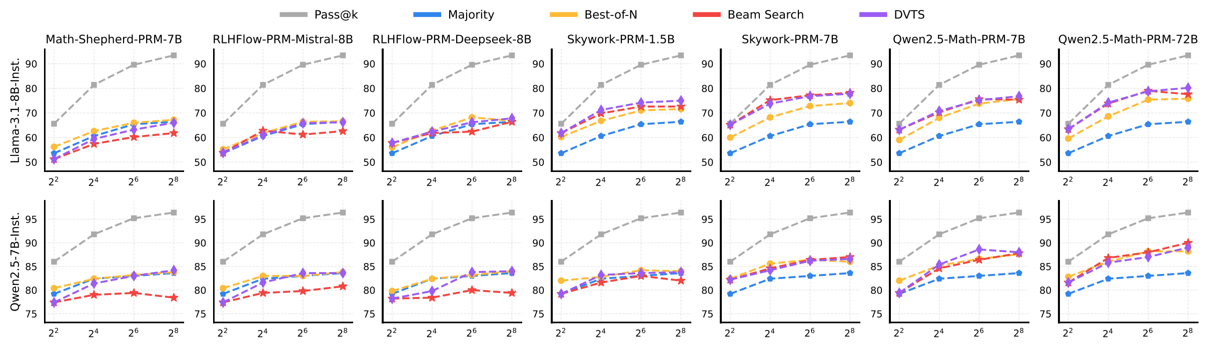

Figure 2: Performance of Llama-3.1-8B-Instruct and Qwen2.5-7B-Instruct on MATH-500 with different PRMs and TTS strategies. The x-axis represents the compute budget.

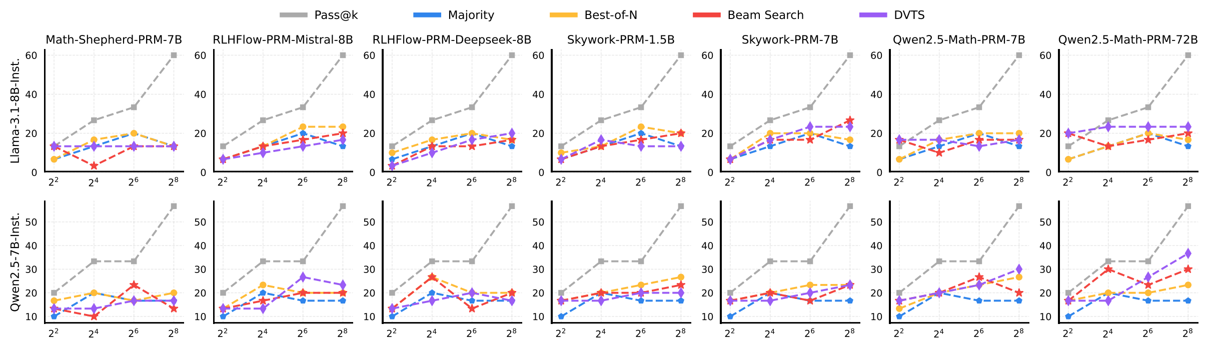

Figure 3: Performance of Llama-3.1-8B-Instruct and Qwen2.5-7B-Instruct on AIME24 with different PRMs and TTS strategies. The x-axis represents the compute budget.

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 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).

4.2 How does TTS improve with different policy models and PRMs? (Q1)

PRMs are hard to generalize across policy models and tasks. As shown in Figure 2, 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 3, 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.

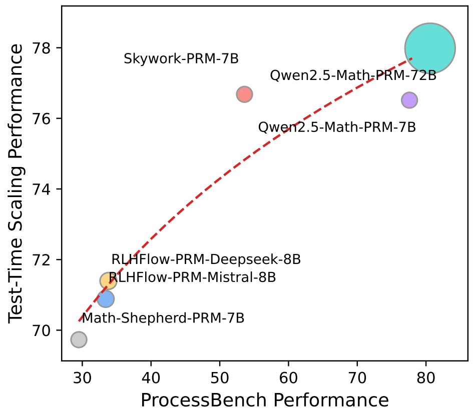

Figure 4: The relationship between TTS performance and process supervision abilities of different PRMs on MATH, where the size of each circle represents the number of parameters of the PRM and the curve represents the fitted function.

The optimal TTS method depends on the PRM used. As shown in Figure 2, 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 4, 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 performance on ProcessBench (Zhang et al., 2025).

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 5 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. (2025), 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., 2025) 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 ($50\% \sim 100\%$), medium ($10\% \sim 50\%$), and hard ($0\% \sim 10\%$).

The optimal TTS methods vary with different difficulty levels. The results in Figure 6 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.

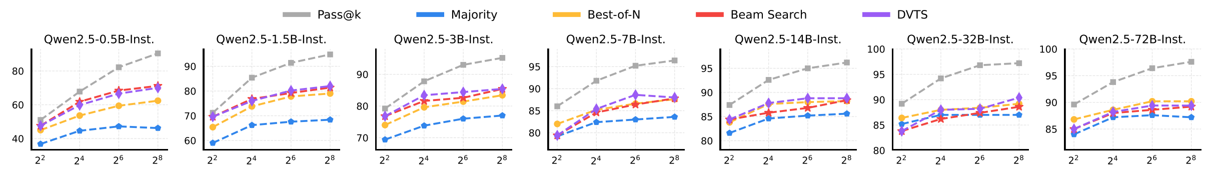

Figure 5: TTS performance of policy models with parameters from 0.5B to 72B on MATH-500 with different scaling methods. The x-axis represents the compute budget.

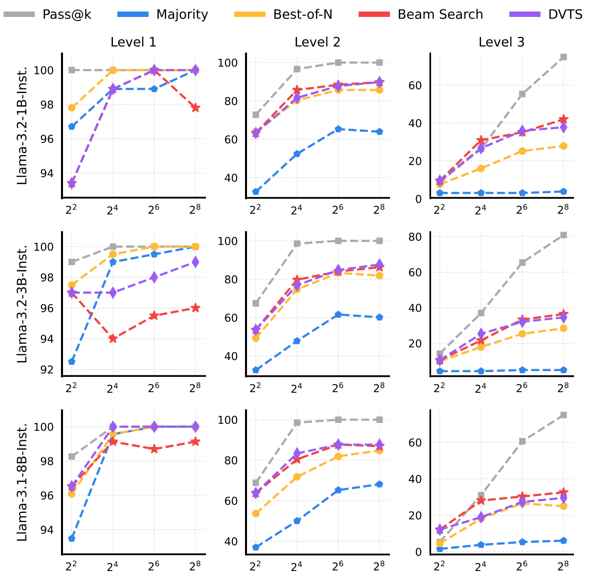

Figure 6: TTS performance of three Llama policy models on MATH-500 with three difficulty levels. The x-axis represents the compute budget.

Takeaways for compute-optimal TTS

- For small policy models (0.5B-3B), use beam search for hard problems and BoN for easy problems.
- For medium policy models (7B-32B), use DVTS for easy and medium problems, and beam search for hard problems.
- For large policy models (72B), use BoN for all problems.
- For tasks that PRMs are Out-Of-Distribution (OOD), use DVTS for 0.5B-7B models and BoN for larger models.

5 Results for Compute-Optimal Test-Time Scaling

With the compute-optimal TTS strategy explored in Section 4, we can summarize the compute-optimal TTS strategy as follows:

In this section, 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?

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[2] 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., 2025; Beeching et al., 2024), we improve the result by **487.0%** (23× → 135×). (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-

[2]For reasoning models, we use majority voting as the TTS method since current PRMs struggle to predict accurate process rewards for reasoning models.

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.**
Proprietary LLMs (CoT)			
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
Open-Source LLMs (CoT)			
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
Open-Source LLMs (Beam Search)			
Llama-3.2-1B-Inst.	65.6	16.7	41.2
Llama-3.2-3B-Inst.	73.2	16.7	45.0
Qwen2.5-0.5B-Inst.	76.0	10.0	43.0
Qwen2.5-1.5B-Inst.	81.6	16.7	49.2
Open-Source LLMs (Compute-Optimal TTS)			
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
DeepSeek-R1-Distill-Qwen-1.5B	91.6	63.3	77.5
DeepSeek-R1-Distill-Qwen-7B	95.2	83.3	89.3

500, but underperforms Llama-3.1-405B-Instruct on AIME24.[3] (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 models can outperform frontier reasoning LLMs with the compute-optimal TTS strategy**. However, if we only use beam search as the TTS method, the performance of Llama-3.2-3B-Instruct still outperforms GPT-4o but is lower than that of Llama-3.1-405B-Instruct, indicating that **compute-optimal TTS further improves the reasoning performance of LLMs beyond merely using beam search**.

[3]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 2: FLOPs comparison between smaller policy models (compute-optimal TTS) and larger ones (CoT).

Policy Model	**Pre-training**	**Inference**	**Total**
Llama-3.2-3B-Inst.	1.62×10^{23}	3.07×10^{17}	1.62×10^{23}
Llama-3.1-405B-Inst.	3.65×10^{25}	4.25×10^{17}	3.65×10^{25}
DeepSeek-R1-Distill-7B	7.56×10^{23}	8.15×10^{17}	7.56×10^{23}
DeepSeek-R1	5.96×10^{25}	4.03×10^{18}	5.96×10^{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. (2025), 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 at most 224×.

5.2 How does compute-optimal TTS improve compared with CoT and majority voting? (Q4)

We aim to analyze the performance gain (with respect to CoT) and efficiency gain (with respect to majority voting) of compute-optimal TTS. The performance gain is defined as the ratio of the performance of compute-optimal TTS to that of CoT, while the efficiency gain is defined as the ratio of the computation efficiency (measured by compute budget N) of compute-optimal TTS to that of majority voting when achieving the same performance.

The results in Table 3 show that scaling test-time compute of small policy models can be 256× more efficient than majority voting and improve reasoning performance by 154.6% over CoT at most. However, as the number of parameters in the policy model increases, the improvement of TTS gradually decreases. These results demonstrate that compute-optimal TTS significantly enhances the reasoning capabilities of LLMs and the TTS is extremely effective for **small policy models**.

Takeaways for compute-optimal TTS

The compute-optimal TTS is more effective for **small models** and can lead to up to 154.6% performance gain compared to CoT and 256× efficiency gain compared to majority voting.

Table 3: Comparison of compute-optimal TTS, CoT, and majority voting with different policy models on MATH-500.

Policy Model	CoT	Major.	C.O. TTS	Perf. Gain	Effi. Gain
Llama-3.2-1B-Inst.	26.0	39.0	66.2	154.6%	>256.0×
Llama-3.2-3B-Inst.	41.4	58.4	78.2	88.9%	14.1×
Llama-3.1-8B-Inst.	49.8	66.4	80.6	61.8%	43.9×
Qwen2.5-0.5B-Inst.	31.6	47.2	76.4	141.8%	>64.0×
Qwen2.5-1.5B-Inst.	54.4	68.4	85.6	57.4%	>256.0×
Qwen2.5-3B-Inst.	64.0	77.0	87.6	36.9%	58.4×
Qwen2.5-7B-Inst.	76.8	83.6	91.0	18.5%	35.9×
Qwen2.5-14B-Inst.	80.2	85.6	91.0	13.5%	51.4×
Qwen2.5-32B-Inst.	82.4	87.0	90.6	10.0%	0.8×
Qwen2.5-72B-Inst.	83.8	87.2	91.8	9.5%	12.9×

6 Related Work

LLM Test-Time Scaling. Scaling LLM test-time compute is an effective way to improve the performance (OpenAI, 2024). Previous works explore majority voting (Wang et al., 2023), search-based methods (Yao et al., 2023; Xie et al., 2023; Khanov et al., 2024; Wan et al., 2024), and refinement (Qu et al., 2024) to improve the performance. For verification-guided test-time compute, Brown et al. (2024) explores inference compute with repeated sampling and domain verifiers, while Kang et al. (2024); Wu et al. (2025); Snell et al. (2025) further explore search-based methods with process reward guidance and Wang et al. (2024c) extends this setting to VLMs. To eliminate the need for external reward models and the generation of extensive samples, Manvi et al. (2024) proposes a self-evaluation method for adaptive and efficient test-time compute. A recent work (Beeching et al., 2024) explores TTS via search methods with diversity. However, these works lack a evaluation with either strong verifiers or policies with different sizes / capabilities. In this paper, we aim to provide a more systematically evaluation with up-to-date policies and verifiers, more challenging tasks, and provide some principles for practical TTS.

Process Reward Models. Previous works show that PRMs are more effective than ORMs (Uesato et al., 2022; Lightman et al., 2024). However, collecting high-quality PRMs data, such as PRM800K (Lightman et al., 2024), is often costly. The researchers explores automatic PRM data collection via direct Monte Carlo estimation (Wang et al., 2024b), detecting relative scores of ORMs (Lu et al., 2024), and efficient MCTS with binary search (Luo et al., 2024). Recently, more advanced PRMs are explored from advantage modeling (Setlur et al., 2025), Q-value rankings (Li and Li, 2025), implicit rewards (Yuan et al., 2024), and entropy regularization (Zhang et al., 2024b) perspectives. Additionally, more open-source PRMs are released (Xiong et al., 2024; Skywork, 2024; Zhang et al., 2024b; Li and Li, 2025; Yuan et al., 2024; Zhang et al., 2025), showing strong performance on mathematical tasks. With the rapid development of PRMs, ProcessBench (Zheng et al., 2024) and PRMBench (Song et al., 2025) are proposed to provide comprehensive evaluation of PRMs. Zhang et al. (2025) provides guidelines for practical development of PRMs.

7 Conclusion & Discussion

In this paper, we present a thorough empirical analysis of compute-optimal test-time scaling from the perspectives of different policy models, PRMs, and more challenging evaluation tasks. Our findings demonstrate the dependency of compute-optimal TTS strategies on policy models, PRMs, and problem difficulty, validating that smaller language models can perform better than larger models when applying compute-optimal TTS. Our results show that a 1B model can achieve better performance than a 405B model through TTS. Additionally, we demonstrate that a 7B PRM can achieve strong TTS results by supervising a more capable 72B policy model, which suggests the importance of investigating a true “weak-to-strong” approach instead of the current “strong-to-weak” supervision for policy optimization. To achieve this goal, we need to develop more efficient supervision methods, as both PRM-based and RL-based approaches have limitations due to their dependence on high-quality supervision. Future work should focus on developing more adaptable and universal supervision mechanisms to boost the performance of small language models on complex tasks and provide new approaches for developing efficient reasoning strategies.

8 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

AI-MO. 2024. Aime 2024.

Anthropic. 2023. Introducing Claude.

Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen Marcus McAleer, Albert Q. Jiang, Jia Deng, Stella Biderman, and Sean Welleck. 2024. Llemma: An open language model for mathematics. In *International Conference on Learning Representations (ICLR)*.

Edward Beeching, Lewis Tunstall, and Sasha Rush. 2024. Scaling test-time compute with open models.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and Azalia Mirhoseini. 2024. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*.

Guoxin Chen, Minpeng Liao, Chengxi Li, and Kai Fan. 2024. Alphamath almost zero: Process supervision without process. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research (TMLR)*.

Ganqu Cui, Lifan Yuan, Zefan Wang, Hanbin Wang, Wendi Li, Bingxiang He, Yuchen Fan, Tianyu Yu, Qixin Xu, Weize Chen, Jiarui Yuan, Huayu Chen, Kaiyan Zhang, Xingtai Lv, Shuo Wang, Yuan Yao, Xu Han, Hao Peng, Yu Cheng, and 4 others. 2025. Process reinforcement through implicit rewards. *arXiv preprint arXiv:2502.01456*.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, and 181 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. PAL: Program-aided language models. In *International Conference on Machine Learning (ICML)*, volume 202, pages 10764–10799.

Zhibin Gou, Zhihong Shao, Yeyun Gong, yelong shen, Yujiu Yang, Minlie Huang, Nan Duan, and Weizhu Chen. 2024. ToRA: A tool-integrated reasoning agent for mathematical problem solving. In *International Conference on Learning Representations (ICLR)*.

Xinyu Guan, Li Lyna Zhang, Yifei Liu, Ning Shang, Youran Sun, Yi Zhu, Fan Yang, and Mao Yang. 2025. rStar-Math: Small llms can master math reasoning with self-evolved deep thinking. *arXiv preprint arXiv:2501.04519*.

Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Ksenia Konyushkova, Lotte Weerts, Abhishek Sharma, Aditya Siddhant, Alex Ahern, Miaosen Wang, Chenjie Gu, and 1 others. 2023. Reinforced self-training (rest) for language modeling. *arXiv preprint arXiv:2308.08998*.

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 *Advances in Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

Arian Hosseini, Xingdi Yuan, Nikolay Malkin, Aaron Courville, Alessandro Sordoni, and Rishabh Agarwal. 2024. V-star: Training verifiers for self-taught reasoners. *arXiv preprint arXiv:2402.06457*.

Zhen Huang, Haoyang Zou, Xuefeng Li, Yixiu Liu, Yuxiang Zheng, Ethan Chern, Shijie Xia, Yiwei Qin, Weizhe Yuan, and Pengfei Liu. 2024. O1 replication journey–part 2: Surpassing o1-preview through simple distillation, big progress or bitter lesson? *arXiv preprint arXiv:2411.16489*.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, and 1 others. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, and 1 others. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Jikun Kang, Xin Zhe Li, Xi Chen, Amirreza Kazemi, Qianyi Sun, Boxing Chen, Dong Li, Xu He, Quan He, Feng Wen, and 1 others. 2024. MindStar: Enhancing math reasoning in pre-trained llms at inference time. *arXiv preprint arXiv:2405.16265*.

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

Kimi. 2024. k0-math.

Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, and 1 others. 2025. Kimi k1.5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, and 1 others. 2024. Training language models to self-correct via reinforcement learning. *arXiv preprint arXiv:2409.12917*.

Joshua Ong Jun Leang, Aryo Pradipta Gema, and Shay B Cohen. 2024. CoMAT: Chain of mathematically annotated thought improves mathematical reasoning. *arXiv preprint arXiv:2410.10336*.

Wendi Li and Yixuan Li. 2025. Process Reward Model with Q-value Rankings. In *International Conference on Learning Representations (ICLR)*.

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 *International Conference on Learning Representations (ICLR)*.

Jianqiao Lu, Zhiyang Dou, WANG Hongru, Zeyu Cao, Jianbo Dai, Yunlong Feng, and Zhijiang Guo. 2024. Autopsv: Automated process-supervised verifier. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Jianguang Lou, Chongyang Tao, Xiubo Geng, Qingwei Lin, Shifeng Chen, and Dongmei Zhang. 2023. Wizardmath: Empowering mathematical reasoning for large language models via reinforced evol-instruct. *arXiv preprint arXiv:2308.09583*.

Liangchen Luo, Yinxiao Liu, Rosanne Liu, Samrat Phatale, Harsh Lara, Yunxuan Li, Lei Shu, Yun Zhu, Lei Meng, Jiao Sun, and 1 others. 2024. Improve mathematical reasoning in language models by automated process supervision. *arXiv preprint arXiv:2406.06592*.

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 *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pages 46534–46594.

Rohin Manvi, Anikait Singh, and Stefano Ermon. 2024. Adaptive inference-time compute: Llms can predict if they can do better, even mid-generation. *arXiv preprint arXiv:2410.02725*.

OpenAI. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

OpenAI. 2024. Learning to reason with llms.

Yiwei Qin, Xuefeng Li, Haoyang Zou, Yixiu Liu, Shijie Xia, Zhen Huang, Yixin Ye, Weizhe Yuan, Hector Liu, Yuanzhi Li, and 1 others. 2024. O1 replication journey: A strategic progress report–part 1. *arXiv preprint arXiv:2410.18982*.

Yuxiao Qu, Tianjun Zhang, Naman Garg, and Aviral Kumar. 2024. Recursive introspection: Teaching language model agents how to self-improve. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Qwen Team. 2024. Qwq: Reflect deeply on the boundaries of the unknown.

Amrith Setlur, Saurabh Garg, Xinyang Geng, Naman Garg, Virginia Smith, and Aviral Kumar. 2024. Rl on incorrect synthetic data scales the efficiency of llm math reasoning by eight-fold. *arXiv preprint arXiv:2406.14532*.

Amrith Setlur, Chirag Nagpal, Adam Fisch, Xinyang Geng, Jacob Eisenstein, Rishabh Agarwal, Alekh Agarwal, Jonathan Berant, and Aviral Kumar. 2025. Rewarding Progress: Scaling Automated Process Verifiers for LLM Reasoning. In *International Conference on Learning Representations (ICLR)*.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, and 1 others. 2024. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Maohao Shen, Guangtao Zeng, Zhenting Qi, Zhang-Wei Hong, Zhenfang Chen, Wei Lu, Gregory Wornell, Subhro Das, David Cox, and Chuang Gan. 2025. Satori: Reinforcement learning with chain-of-action-thought enhances llm reasoning via autoregressive search. *arXiv preprint arXiv:2502.02508*.

Skywork. 2024. Skywork-o1.

Skywork o1 Team. 2024. Skywork-o1 open series. `https://huggingface.co/Skywork`.

Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2025. Scaling Test-Time Compute Optimally Can be More Effective than Scaling LLM Parameters. In *International Conference on Learning Representations (ICLR)*.

Mingyang Song, Zhaochen Su, Xiaoye Qu, Jiawei Zhou, and Yu Cheng. 2025. Prmbench: A fine-grained and challenging benchmark for process-level reward models. *arXiv preprint arXiv:2501.03124*.

Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

Zhengyang Tang, Xingxing Zhang, Benyou Wang, and Furu Wei. 2024. MathScale: Scaling instruction tuning for mathematical reasoning. In *International Conference on Machine Learning (ICML)*, volume 235, pages 47885–47900.

Yuxuan Tong, Xiwen Zhang, Rui Wang, Ruidong Wu, and Junxian He. 2024. DART-math: Difficulty-aware rejection tuning for mathematical problem-solving. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, and 1 others. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Luong Trung, Xinbo Zhang, Zhanming Jie, Peng Sun, Xiaoran Jin, and Hang Li. 2024. Reft: Reasoning with reinforced fine-tuning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7601–7614.

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

Ziyu Wan, Xidong Feng, Muning Wen, Stephen Marcus Mcaleer, Ying Wen, Weinan Zhang, and Jun Wang. 2024. AlphaZero-like tree-search can guide large language model decoding and training. In *International Conference on Machine Learning (ICML)*, volume 235, pages 49890–49920.

Jun Wang, Meng Fang, Ziyu Wan, Muning Wen, Jiachen Zhu, Anjie Liu, Ziqin Gong, Yan Song, Lei Chen, Lionel M Ni, and 1 others. 2024a. Openr: An open source framework for advanced reasoning with large language models. *arXiv preprint arXiv:2410.09671*.

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

Xiyao Wang, Zhengyuan Yang, Linjie Li, Hongjin Lu, Yuancheng Xu, Chung-Ching Lin, Kevin Lin, Furong Huang, and Lijuan Wang. 2024c. Scaling inference-time search with vision value model for improved visual comprehension. *arXiv preprint arXiv:2412.03704*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations (ICLR)*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in neural information processing systems (NeurIPS)*, volume 35, pages 24824–24837.

Yixuan Weng, Minjun Zhu, Fei Xia, Bin Li, Shizhu He, Shengping Liu, Bin Sun, Kang Liu, and Jun Zhao. 2023. Large language models are better reasoners with self-verification. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 2550–2575.

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. 2025. Inference Scaling Laws: An Empirical Analysis of Compute-Optimal Inference for LLM Problem-Solving. In *International Conference on Learning Representations (ICLR)*.

Yuxi Xie, Kenji Kawaguchi, Yiran Zhao, James Xu Zhao, Min-Yen Kan, Junxian He, and Michael Xie. 2023. Self-evaluation guided beam search for reasoning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pages 41618–41650.

Wei Xiong, Hanning Zhang, Nan Jiang, and Tong Zhang. 2024. An implementation of generative prm. https://github.com/RLHFlow/RLHF-Reward-Modeling.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, and 40 others. 2024a. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, and 22 others. 2024b. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.

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

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pages 11809–11822.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng YU, Zhengying Liu, Yu Zhang, James Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. 2024. MetaMath: Bootstrap your own mathematical questions for large language models. In *International Conference on Learning Representations (ICLR)*.

Lifan Yuan, Wendi Li, Huayu Chen, Ganqu Cui, Ning Ding, Kaiyan Zhang, Bowen Zhou, Zhiyuan Liu, and Hao Peng. 2024. Free process rewards without process labels. *arXiv preprint arXiv:2412.01981*.

Eric Zelikman, Georges Raif Harik, Yijia Shao, Varuna Jayasiri, Nick Haber, and Noah Goodman. 2024. Quiet-STar: Language models can teach themselves to think before speaking. In *Conference on Language Modeling (COLM)*.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. 2022. STaR: Bootstrapping reasoning with reasoning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pages 15476–15488.

Liang Zeng, Liangjun Zhong, Liang Zhao, Tianwen Wei, Liu Yang, Jujie He, Cheng Cheng, Rui Hu, Yang Liu, Shuicheng Yan, and 1 others. 2024. Skywork-Math: Data scaling laws for mathematical reasoning in large language models–the story goes on. *arXiv preprint arXiv:2407.08348*.

Weihao Zeng, Yuzhen Huang, Wei Liu, Keqing He, Qian Liu, Zejun Ma, and Junxian He. 2025. 7b model and 8k examples: Emerging reasoning with reinforcement learning is both effective and efficient. https:

//hkust-nlp.notion.site/simplerl-reason. Notion Blog.

Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. 2024a. ReST-MCTS*: LLM self-training via process reward guided tree search. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Hanning Zhang, Pengcheng Wang, Shizhe Diao, Yong Lin, Rui Pan, Hanze Dong, Dylan Zhang, Pavlo Molchanov, and Tong Zhang. 2024b. Entropy-regularized process reward model. *arXiv preprint arXiv:2412.11006*.

Zhenru Zhang, Chujie Zheng, Yangzhen Wu, Beichen Zhang, Runji Lin, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. 2025. The lessons of developing process reward models in mathematical reasoning. *arXiv preprint arXiv:2501.07301*.

Yu Zhao, Huifeng Yin, Bo Zeng, Hao Wang, Tianqi Shi, Chenyang Lyu, Longyue Wang, Weihua Luo, and Kaifu Zhang. 2024. Marco-o1: Towards open reasoning models for open-ended solutions. *arXiv preprint arXiv:2411.14405*.

Chujie Zheng, Zhenru Zhang, Beichen Zhang, Runji Lin, Keming Lu, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. 2024. Processbench: Identifying process errors in mathematical reasoning. *arXiv preprint arXiv:2412.06559*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, and 1 others. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pages 46595–46623.

A Test-Time Scaling Methods

In this paper, we use Best-of-N, Beam Search and Diverse Verifier Tree Search (DVTS) as the test-time scaling methods. The used TTS methods are shown in Figure 7.

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 `<EOS>` 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.

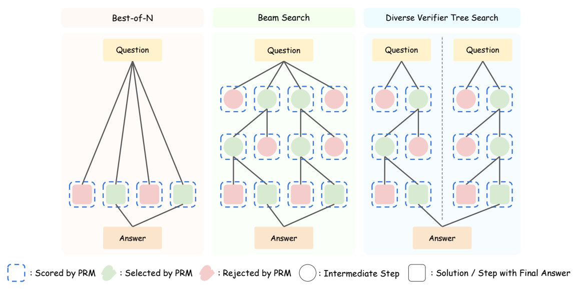

Figure 7: Comparison of different external TTS methods.

B More Discussion on Relate Work

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

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

C Experimental Details

Scoring and Voting Methods. Following Wang et al. (2024a), we consider three scoring methods: *PRM-Min*, *PRM-Last*, and *PRM-Avg*, and three voting methods: *Majority Vote*, *PRM-Max*, and *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., 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., score $= \mathcal{R}_H$. (3) *PRM-Avg* scores each trajectory by the average reward among all steps, i.e., 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[4], 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. All methods are evaluated using a single run.

The performance gain in Table 3 is computed as: Performance Gain $= \left(\frac{\text{Performance of compute-optimal TTS}}{\text{Performance of CoT}} - 1\right) \times 100\%$. The efficiency gain in Table 3 is computed by comparing the compute budgets required to achieve the same performance using compute-optimal TTS and majority voting, following Snell et al. (2025). Specifically:

1. Identify the highest performance point on the majority voting curve.

2. Find the corresponding compute budget on the compute-optimal TTS curve that achieves the same performance (using linear interpolation if necessary).

3. Compute the efficiency gain as: Efficiency Gain $= \left(\frac{\text{Compute budget of majority voting}}{\text{Compute budget of compute-optimal TTS}}\right)$.

MATH-500 (Lightman et al., 2024) is under the MIT License and AIME24 (AI-MO, 2024) is under Apache license 2.0.

[4]https://github.com/openreasoner/openr

D Additional Experimental Results

D.1 Does PRMs have bias towards specific response lengths or sensitivity to voting methods?

Table 4: 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

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 singificantly. 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[5] and RLHFlow-PRM-Deepseek-8B[6]. As shown in Table 4, 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.

Table 5: Performance of TTS with different voting methods on MATH-500.

	Skywork-PRM-7B	Qwen2.5-Math-PRM-7B
Majority Vote	86.8	87.6
PRM-Min-Max	83.0	87.4
PRM-Min-Vote	86.6	87.6
PRM-Last-Max	84.4	87.6
PRM-Last-Vote	87.0	87.6
PRM-Avg-Max	85.8	87.8
PRM-Avg-Vote	86.8	87.6

[5]https://huggingface.co/datasets/RLHFlow/Mistral-PRM-Data

[6]https://huggingface.co/datasets/RLHFlow/Deepseek-PRM-Data

PRMs are sensitive to voting methods. From the results in Table 5, it is shown that Skywork-PRM-7B works better with *PRM-Vote* than with *PRM-Max*, while Qwen2.5-Math-PRM-7B is not very 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.

D.2 Is TTS more effective than long-CoT-based methods?

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 learning. (2) **Eurus-2** (Cui et al., 2025): This method enhances the reasoning abilities of LLMs through implicit process rewards and online RL. (3) **SimpleRL** (Zeng et al., 2025): This method replicates self-reflection with only 8K training data. (4) **Satori** (Shen et al., 2025): This method first learn the format and then improves the reasoning abilities via RL. (5) **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 6, we find that TTS with Qwen2.5-7B-Instruct outperforms rStar-Math, Eurus-2, SimpleRL, and Satori 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 RL or self-play methods but is less effective than distilling from strong reasoning models. Also, TTS is more effective on simpler tasks than on more complex tasks.

Table 6: Comparison of compute-optimal TTS with long-CoT methods on MATH-500 and AIME24.

Policy Model	**MATH-500**	**AIME24**	**Avg.**
Open-Source LLMs (CoT)			
Qwen2.5-7B-Inst.	76.8	13.3	45.1
Qwen2.5-Math-7B-Inst.	79.8	13.3	46.6
Long-CoT Methods (CoT)			
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
Satori-Qwen-7B	83.6	23.3	53.5
DeepSeek-R1-Distill-Qwen-7B	92.4	63.3	77.9
Open-Source LLMs (TTS)			
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

D.3 Full Results of Test-Time Scaling with Different Policy Models, PRMs, and Scaling Methods

The full results of TTS with different policy models, scaling methods, and problem difficulty levels are shown in Figure 8. The full results of TTS with different policy models, PRMs, and scaling methods are shown in Figure 9 and Figure 10.

E 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 7 and Table 8, respectively. Following Beeching et al. (2024), we use the system prompt of the official evaluation[7] for Llama 3 to prevent performance drop.

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 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

[7]`https://huggingface.co/datasets/meta-llama/Llama-3.2-1B-Instruct-evals`

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 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.

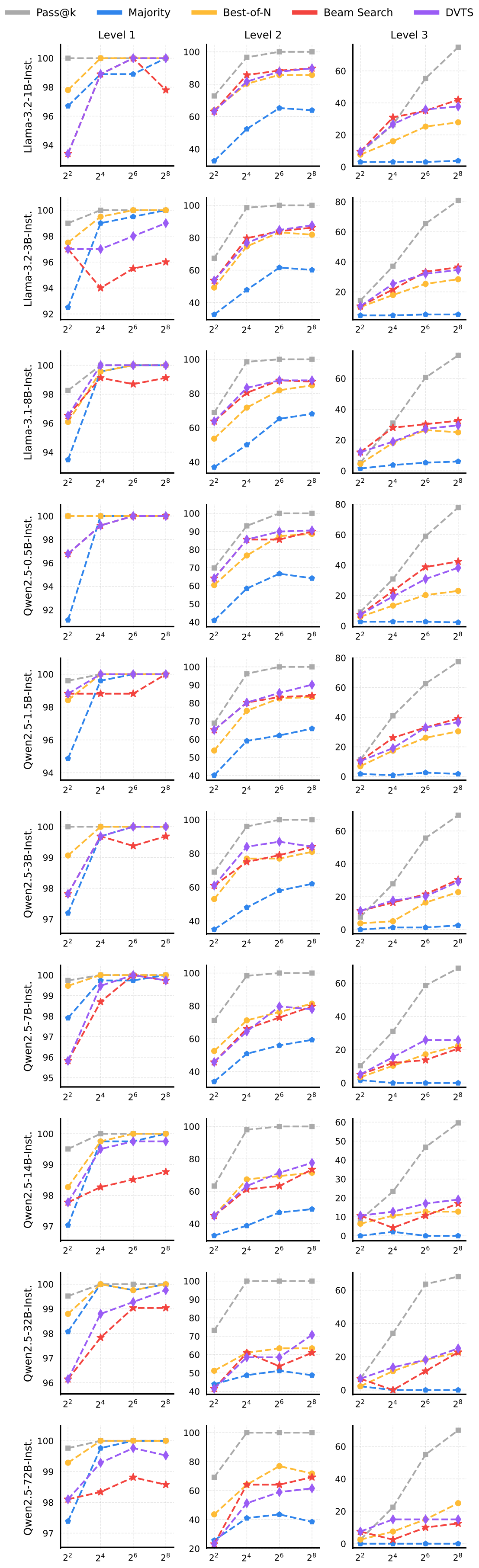

Figure 8: TTS performance of three Llama policy models on MATH-500 with different difficulty levels.

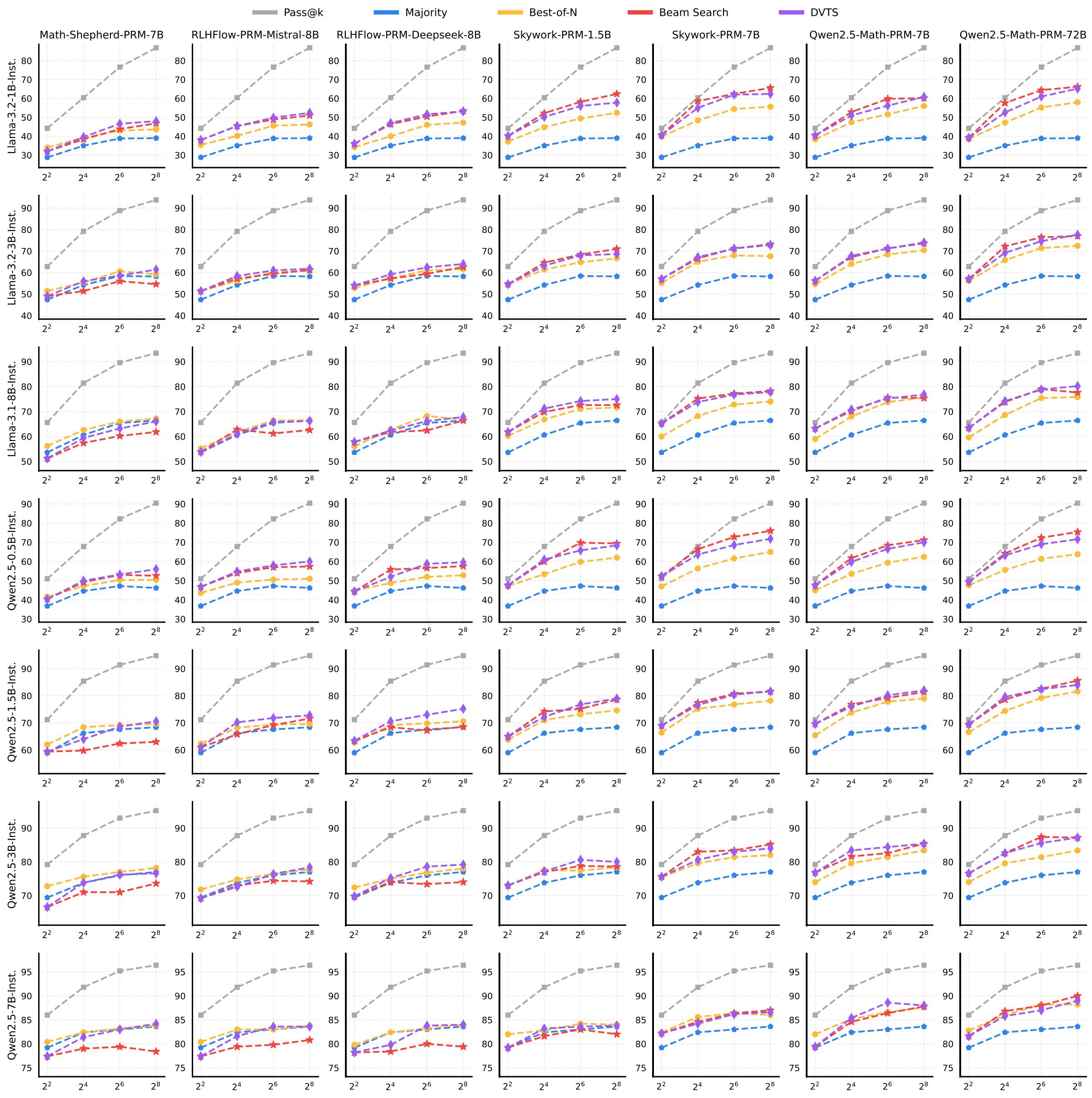

Figure 9: TTS performance of different policy models on MATH-500 with different PRMs and scaling strategies.

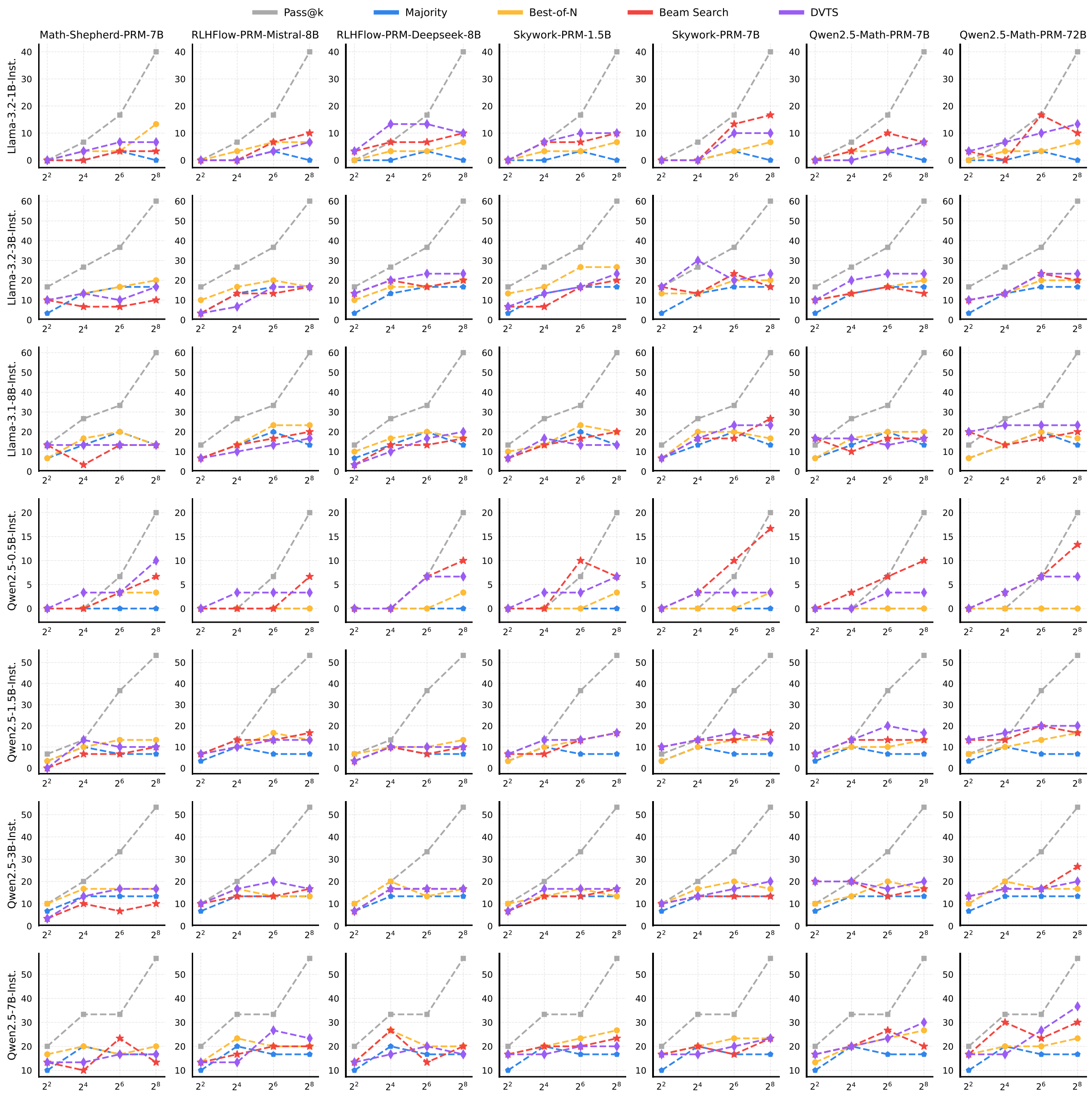

Figure 10: TTS performance of different policy models on AIME24 with different PRMs and scaling strategies.

Table 7: 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: $\boxed{answer}$. I hope it is correct.

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

Table 8: System prompt for Qwen2.5 series models.

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

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. \nSince 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.\nThe 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.\nThe number 220 is the smallest multiple of 30 that can be formed with the digits 0 and 2. ки\n [score=0.56]

Step 4: Multiplying 220 by 3 results in 660, which is a multiple of 30.\nWe'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\nTo 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\nFor 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\nFor 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\nThe 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\nThe 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\nThe 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\nThe 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\nThe 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\nInstead 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\nThe 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.

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.\n - 242 is even, so it is divisible by 2.\n \\[\n 242 \\div 2 = 121\n \\]\n So, \\(242 = 2 \\times 121\\).\n - Next, we factor 121. We know that 121 is \\(11 \\times 11\\).\n \\[\n 121 = 11 \\times 11\n \\]\n 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.\n \\[\n \\sqrt{242} = \\sqrt{2 \\times 11 \\times 11}\n \\] [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}\\).\n \\[\n \\sqrt{242} = \\sqrt{2 \\times 11 \\times 11} = \\sqrt{2} \\times \\sqrt{11 \\times 11} = \\sqrt{2} \\times 11 = 11\\sqrt{2}\n \\] [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.

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));\n label(\"D\",D,NE);\n label(\"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:**\n - We are given \\(\\sin D = 0.7\\).\n - \\(\\angle DEF \\) is a right angle.\n - \\(EF = 7 \\). [score=0.99] ✓

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

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

4. **Solve for \\(DE \\):**\n - To isolate \\(DE \\), multiply both sides of the equation by \\(DE \\):\n \\[\n 0.7 \\cdot DE = 7\n \\]\n - Divide both sides by 0.7:\n \\[\n DE = \\frac{7}{0.7}\n \\]\n - Simplify the division:\n \\[\n DE = 10\n \\] [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));\n label(\"D\",D,NE);\n label(\"E\",E,NW);\n label(\"F\",F,SW);\n label(\"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:\n\\[\\sin D = \\frac{EF}{DE} = 0.7 \\]\nSince \\(EF = 7 \\), we substitute to get:\n\\[\\frac{7}{DE} = 0.7 \\]\nTo find \\(DE \\), we solve the equation:\n\\[DE = \\frac{7}{0.7} = \\frac{7}{\\frac{7}{10}} = 10 \\] [score=0.88] ✗

Thus, the length of \\(DE \\) is:\n\\[\\boxed{10} \\] [score=0.82] ✗

Figure 14: TTS case of Error Neglect.

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:\n\\[\n\\frac{BD}{DC} = \\frac{AB}{AC} = \\frac{5}{10} = \\frac{1}{2}\n\\]\nSince \\(BD = DC\\), we can use the Power of a Point theorem at \\(D\\). The power of point \\(D\\) with respect to the circle is:\n\\[\nDB^2 = DC^2 = DA \\cdot DP\n\\]\nSince \\(D\\) is the intersection of the tangents at \\(B\\) and \\(C\\), we have:\n\\[\nDB^2 = DC^2 = BD^2 = BD \\cdot BP\n\\]\nLet \\(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:\n\\[\nBD^2 = DA \\cdot DP \\implies \\left(\\frac{9}{2}\\right)^2 = DA \\cdot DP \\implies \\frac{81}{4} = DA \\cdot DP\n\\]\nWe 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:\n\\[\nAD = \\frac{2 \\cdot AB \\cdot AC \\cdot \\cos \\left(\\frac{A}{2}\\right)}{AB + AC}\n\\]\nHowever, 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\\):\n\\[\nAP = \\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}\n\\]\nThus, \\(AP = \\frac{117}{40}\\), and the sum \\(m + n\\) is:\n\\[\n117 + 40 = 157\n\\]\nTherefore, the answer is:\n\\[\n\\boxed{157}\n\\] **[score=0.92]** ✗

Figure 15: TTS case of Error Localization Bias.

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:\n\\[7n + 53 \\equiv 0 \\pmod{8} \\]\n\\[7n \\equiv -53 \\pmod{8} \\]\nSince \\(-53 \\equiv -53 + 64 = 11 \\equiv 3 \\pmod{8}\\), we have:\n\\[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:\n\\[7x \\equiv 1 \\pmod{8} \\]\nBy testing values, we find that \\(x = 7 \\) works because:\n\\[7 \\cdot 7 = 49 \\equiv 1 \\pmod{8} \\]\nThus, multiplying both sides of \\(7n \\equiv 3 \\pmod{8} \\) by 7, we get:\n\\[n \\equiv 21 \\pmod{8} \\]\n\\[n \\equiv 5 \\pmod{8} \\]\nSo, \\(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:\n\\[56k + 88 > 200 \\]\n\\[56k > 112 \\]\n\\[k > 2 \\]\nThe smallest integer \\(k \\) that satisfies this inequality is \\(k = 3 \\). Substituting \\(k = 3 \\) into the expression for the total number of coins, we get:\n\\[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:\n\\[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.

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:\n\\[7x + 53 \\equiv 0 \\pmod{8} \\]\nSince \\(7 \\equiv -1 \\pmod{8} \\), we can rewrite the equation as:\n\\[-x + 53 \\equiv 0 \\pmod{8} \\]\n\\[-x \\equiv -53 \\pmod{8} \\]\n\\[x \\equiv 53 \\pmod{8} \\]\n\\[x \\equiv 5 \\pmod{8} \\]\nThis means that \\(x \\) can be written in the form:\n\\[x = 8k + 5 \\]\nfor some integer \\(k \\). [score=0.90] ✓

Next, we substitute \\(x = 8k + 5 \\) into the expression for the total number of coins:\n\\[7x + 53 = 7(8k + 5) + 53 = 56k + 35 + 53 = 56k + 88 \\]\nWe need \\(56k + 88 \\) to be more than 200:\n\\[56k + 88 > 200 \\]\n\\[56k > 112 \\]\n\\[k > 2 \\]\nThe smallest integer \\(k \\) that satisfies this inequality is \\(k = 3 \\). Substituting \\(k = 3 \\) into the expression for the total number of coins, we get:\n\\[56 \\cdot 3 + 88 = 168 + 88 = 256 \\]\nThus, the smallest number of coins you could have had before finding the bag of 53 coins is:\n\\[7x = 7(8 \\cdot 3 + 5) = 7 \\cdot 29 = 203 \\]\nTherefore, 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.