Step-level Verifier-guided Hybrid Test-Time Scaling for Large Language Models

Kaiyan Chang[1], Yonghao Shi[1], Chenglong Wang[1], Hang Zhou[1], Chi Hu[3], Xiaoqian Liu[1], Yingfeng Luo[1], Yuan Ge[1], Tong Xiao[1,2*], and Jingbo Zhu[1,2]

[1] School of Computer Science and Engineering, Northeastern University, Shenyang, China

[2] NiuTrans Research, Shenyang, China

[3] ByteDance, Hangzhou, China

changkaiyan_neu@outlook.com,
{xiaotong, zhujingbo}@mail.neu.edu.cn

Abstract

Test-Time Scaling (TTS) is a promising approach to progressively elicit the model's intelligence during inference. Recently, training-based TTS methods, such as continued reinforcement learning (RL), have further surged in popularity, while training-free TTS methods are gradually fading from prominence. However, the additional computation overhead of training amplifies the burden on test-time scaling. In this paper, we focus on training-free TTS methods for reasoning. We first design *Conditional Step-level Self-refinement*, a fine-grained sequential scaling method guided by process verification. On top of its effectiveness, we further combine it with other classical parallel scaling methods at the step level, to introduce a novel inference paradigm called *Hybrid Test-Time Scaling*[1]. Extensive experiments on five instruction-tuned LLMs across different scales (3B-14B) and families demonstrate that hybrid strategy incorporating various training-free TTS methods at a fine granularity has considerable potential for expanding the reasoning performance boundaries of LLMs.

1 Introduction

The remarkable breakthroughs of large language models (LLMs) on complex reasoning tasks (Li et al., 2025; Xiao and Zhu, 2025; Fan et al., 2025), such as OpenAI-o1 (OpenAI, 2024) and DeepSeek-R1 (DeepSeek-AI et al., 2025), are largely attributed to extensive inference-time computation for deep thinking, a paradigm known as "Test-Time Scaling" (TTS) (Snell et al., 2024). Current TTS methods can be broadly categorized into two groups based on training requirements: Training-based methods continue training LLMs to generate longer chain-of-thoughts (CoT) (Wei et al., 2022), including reinforcement learning (RL) (Shao et al., 2024; Yu et al., 2025; Zheng et al., 2025) and supervised fine-tuning (SFT) (Morishita et al., 2024; Muennighoff et al., 2025). Training-free methods guide LLMs to systematically explore the solution space, including parallel scaling (Brown et al., 2024; Qi et al., 2025) and sequential scaling (Gou et al., 2023; Zhang et al., 2024b).

Test-Time Scaling Methods	Additional Training	GPQA Diamond
Hybrid Test-Time Scaling (Ours)	×	**51.5**
Continued Reinforcement Learning	✓	49.1

Table 1: Reasoning performance activated by training-free and training-based TTS methods.

Although training-based TTS methods have shown outstanding reasoning performance, especially RL, they incur not only additional train-time computational cost but also unnecessary test-time computational overhead by "overthinking" simple problems (Chen et al., 2024). In contrast, training-free TTS methods can effectively mitigate these issues. They typically control the generation process through verification (Ji et al., 2025; Chen et al., 2025), which guides the LLM toward correct scaling directions and stops scaling once a valid solution is confirmed. Notably, the computational overhead of verification is significantly lower than that of generation, making verifier-guided TTS advantageous for its low operational cost and "plug-and-play" flexibility. However, training-free TTS methods generally underperform on complex reasoning tasks, particularly evident in sequential scaling like self-refinement (Madaan et al., 2023; Huang et al., 2024) which yields marginal performance gains.

In this paper, we explore how to effectively integrate various training-free TTS methods to further unlock the reasoning potential of LLMs. To this end, we first propose *Conditional Step-level Self-refinement* to validate the effectiveness of fine-grained sequential scaling. On this basis, we further introduce *Step-level Verifier-guided Hybrid*

* Corresponding author.

[1] https://github.com/Lucky-259/Hybrid_TTS

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 18462–18477

Test-Time Scaling as shown in Figure 1, which combines parallel (Best-of-N) and sequential (self-refinement) scaling within a step-level tree search.

Extensive experiments on MATH500, AIME24, and GPQA Diamond datasets show that Hybrid Test-Time Scaling consistently enhances the reasoning performance of five instruction-tuned LLMs across various scales (3-14B) and families (Qwen2.5, LLaMA3.1, Gemma3). Guided by the most capable process reward model (Qwen2.5-Math-PRM-7B), it achieves a maximum performance increase of 28.6%. Surprisingly, our method empowers the Qwen2.5-3B-Instruct model to surpass an RL-enhanced DeepSeek-R1-Distill-Qwen-7B model by 2.4% on the GPQA Diamond dataset (Table 1). These findings suggest that the essence of test-time scaling lies in the extensive exploration and precise exploitation of the LLM's solution space. Our primary contributions are as follows:

- To the best of our knowledge, we are the first to propose conditional step-level self-refinement and demonstrate the potential of fine-grained sequential scaling.
- We introduce Hybrid Test-Time Scaling, a novel inference paradigm that combines parallel and sequential scaling at the step level, leveraging a powerful verifier to efficiently navigate the model's solution space.
- We conduct a comprehensive empirical evaluation across diverse LLMs and challenging reasoning benchmarks. The consistent performance improvements not only illustrate the versatility and effectiveness of our hybrid strategy but also provide valuable insights into the integration of different training-free TTS methods.

2 Related Works

2.1 Training-free Test-Time Scaling

Training-free TTS methods fall into two categories: Parallel Scaling samples multiple responses in parallel and aggregates them into a final answer, while Sequential Scaling iteratively refines an initial response. Hybrid Scaling leverages the complementary benefits of these two paradigms, mirroring the human cognitive process of first generating diverse hypotheses (divergent thinking) and then systematically refining them toward an optimal solution (convergent thinking) (Zhang et al., 2025c).

Prior work has explored the integration of different training-free TTS methods. For instance, Kang et al. (2024); Liu et al. (2025); Wu et al. (2024b) introduce the core idea of Best-of-N into the search process to improve inference efficiency. However, their explorations remain confined to the parallel scaling. Other research explores hybrid scaling, typically integrating sequential scaling into a structured search framework to achieve the exploration-exploitation trade-off in the solution space:

- **Solution-level.** One line of work considers a complete solution as a node in Monte Carlo Tree Search (MCTS) (Zhang et al., 2024a; Rabby et al., 2024; Zhang et al., 2025a), where iterative self-refinement acts as the simulation process. Nevertheless, such solution-level exploration remains insufficient for complex reasoning.
- **Step-level.** Another line of work treats each reasoning step as a node. Representative methods like Tree-of-Thought (ToT) (Yao et al., 2023) and Graph-of-Thought (GoT) (Besta et al., 2024) construct a network (tree or arbitrary graph) of thoughts and prompt the LLM itself to evaluate and refine incorrect steps. SR-MCTS (Zhao et al., 2024) further refines this to mini-steps (32/64 tokens), using the log probabilities of alternative tokens to evaluate node values. Despite the finer granularity of step-level exploration, the self-feedback mechanism is susceptible to the model's inherent hallucinations, thereby limiting its overall reasoning performance.

Therefore, we aim to introduce a high-quality step-level verifier to provide precise and real-time evaluations of intermediate reasoning steps, guiding the LLM to prevent error accumulation and scale toward the correct solution efficiently.

2.2 Verifiers

The verification mechanism plays a crucial role in TTS. It can be used to select the optimal solution from candidate responses in parallel scaling and determine when to stop generation in sequential scaling (Zhang et al., 2025c). One of the most common forms of verification is the LLM-based generative reward model (Zhang et al., 2025b; Wang et al., 2025). Two types of verifiers have emerged based on whether they evaluate the final answer or the reasoning path: outcome reward models (ORMs) and process reward models (PRMs).

To investigate the efficacy of fine-grained TTS, we focus on process-level verifiers capable of accurately evaluating intermediate reasoning steps. The concept of process verification was first introduced by DeepMind (Uesato et al., 2022), laying the foundation for subsequent research. OpenAI (Lightman et al., 2023) later trains a reward model on a larger-scale, human-annotated dataset PRM800K, demonstrating that PRMs outperform ORMs in evaluating reasoning quality. Since then, research on PRM training has become a prominent area of research. For instance, Math-Shepherd (Wang et al., 2023a) employs automatically constructed process-wise supervision data instead of manual annotation. Zhang et al. (2025d) propose a consensus filtering mechanism that combines MC estimation with LLM-as-a-judge to improve both model performance and data efficiency in PRM training.

In this paper, the verifier's feedback dictates the scaling direction of each TTS method. Therefore, it is natural to select a verifier capable of providing accurate step-level verification. According to PRM-Bench (Song et al., 2025), a comprehensive benchmark for PRMs, Qwen2.5-Math-PRM-7B (Zhang et al., 2025d) is currently the most capable PRM among those with 7B/8B parameters. Thus, we adopt it as our step-level verifier to investigate fine-grained Hybrid Test-Time Scaling.

3 Method

3.1 Control of Generation Granularity

Controlling the LLM generation granularity at the step level is a precondition for fine-grained training-free TTS. We observe that instruction-tuned open-source LLMs (*e.g.* Qwen, LLaMA, and Gemma) naturally segment reasoning steps using "`\n\n`" when performing reasoning tasks. We thus implement a "Pause-then-Continue" strategy to control their generation granularity. Specifically, we set "`\n\n`" as a delimiter that signals the model to pause after generating each reasoning step. We then append "`\n`" to the context to prompt the model to resume generation until a terminator token "`<eos>`". This "Pause-then-Continue" strategy requires LLMs to have strong instruction-following and reasoning capabilities. Therefore, we utilize instruction-tuned LLMs with at least 3B parameters to investigate fine-grained TTS.

3.2 Conditional Step-level Self-refinement

Self-refinement is a representative sequential scaling method, where meta-prompts guide the LLM to automatically perform feedback and revision on the initial solution. However, its improvements in reasoning performance are significantly lower than other common parallel scaling methods. We hypothesize that the scaling granularity of solution-level self-refinement is too coarse-grained, leading to inaccurate identification of erroneous steps or unnecessary modifications of correct ones. Therefore, we propose a conditional step-level self-refinement method for fine-grained sequential scaling, where the verification process plays a critical role.

Why verify? We focus on self-refinement for reasoning tasks. The LLM first generates a solution based on a CoT prompt, then reflects on the weaknesses with a Critic prompt, and finally revises the original solution based on the question, current solution, and reflection guided by a Rewrite prompt. In fact, the above reflection and revision process is often conducted unconditionally, which introduces two major concerns. Firstly, forcing self-reflection even when the solution is already correct can cause the model to second-guess its own valid reasoning, introducing the risk of erroneously correcting a valid answer and incurring unnecessary refinement overhead. Secondly, in the absence of a clear stopping signal, the LLM may overlook an already optimal solution and continue making unnecessary or even detrimental modifications. Hence, it is crucial to ensure the correctness of the reasoning process through accurate and real-time verification for superior performance.

What to verify? Step-by-step CoT reasoning is sequential and prone to error accumulation, as early-stage errors can often lead to incorrect final answers. Therefore, we aim to verify the correctness of the logical relationship between the current step and the preceding context, thereby ensuring the quality of intermediate reasoning steps. Importantly, the verification of the current step takes place before the generation of the next step, a design that yields two benefits: (i) it ensures subsequent steps are generated based on verified high-quality context, thereby mitigating the typical error accumulation in long CoT; and (ii) it prevents unnecessary refinement of already correct steps, reducing the likelihood of over-refinement or hallucination-induced overthinking.

How to verify? The PRM is a fine-grained verifier that scores each reasoning step on a scale of 0 to 1 based on the question and preceding steps. We thus leverage the reliable evaluation capabilities of the PRM to design our conditional step-level self-

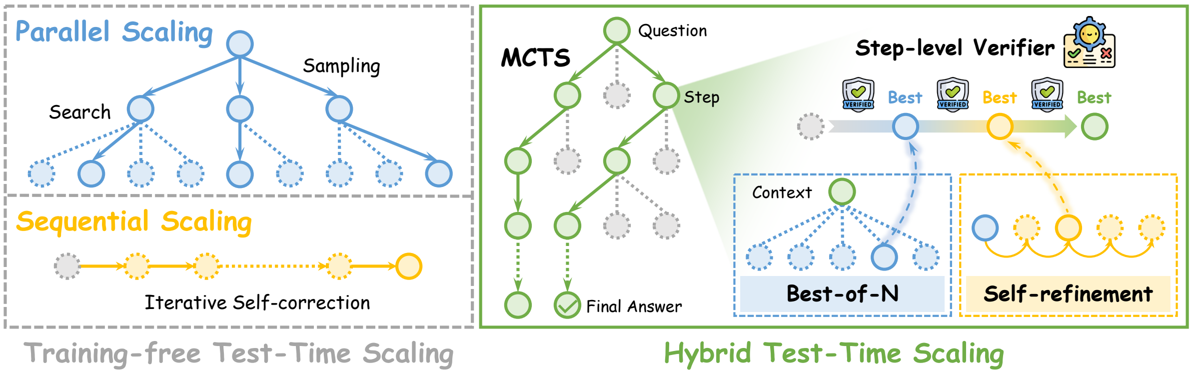

Figure 1: **Left**: The two mainstream paradigms of training-free TTS, parallel and sequential scaling. **Right**: Our proposed Step-level Verifier-guided Hybrid Test-Time Scaling, which incorporates representative methods from both paradigms. Specifically, it performs "Deep Scaling" on each reasoning step within an MCTS framework through a two-stage screening process: (1) Multiple candidates are generated via Best-of-N sampling and the best step is selected using the PUCT algorithm. (2) This candidate undergoes a PRM-guided conditional self-refinement to produce the final best step, completing the "*best-of-the-best*" selection cycle.

refinement strategy with the following principles:

- **Whether to reflect.** Reflection is triggered when the PRM score for the current step falls below a predefined threshold, indicating low confidence in its correctness.
- **How to update.** The reasoning step is updated only if the PRM score of the revised step exceeds that of the current step; otherwise, the current step is retained.
- **When to stop.** Refinement stops when the PRM score surpasses a target threshold, when incremental improvements plateau, or when a maximum number of iterations is reached.

A key property of our proposed conditional step-level self-refinement is its orthogonality to other training-free TTS methods. To further balance performance and efficiency, we conducted a series of preliminary experiments to identify the optimal conditions for improved step-level self-refinement, as discussed in Section 5.1.

3.3 Step-level Verifier-guided Hybrid Test-Time Scaling

Inspired by the effectiveness of step-level self-refinement, we further investigate the potential of training-free test-time scaling at a finer granularity. Specifically, we aim to integrate representative techniques in both parallel and sequential scaling within a unified step-level framework. As illustrated on the right side of Figure 1, our Hybrid Test-Time Scaling paradigm synergistically combines Best-of-N (BoN) and self-refinement in a tree search framework at the step level, all guided by a PRM that serves as the step-level verifier to ensure the quality of each intermediate reasoning step.

In concrete terms, we construct a reasoning tree where the initial question serves as the root node, and each subsequent node represents a reasoning step. Our algorithm 1 for step-level Hybrid Test-time Scaling (detailed in Appendix B) employs the PRM to evaluate the value of each candidate step. The most promising step with the highest score is then selected for expansion as the next node in the reasoning path. In this way, an optimal reasoning path is progressively constructed, ultimately forming a complete solution. The three-stage pipeline for scaling each step is as follows:

1. **Best-of-N.** At each step of the tree search, we begin from the current node and perform N parallel samplings of candidate reasoning steps. To balance exploration and exploitation, we select the best action a by maximizing a score adapted from the Probabilistic Upper Confidence Tree (PUCT) (Schrittwieser et al., 2020) as defined in Eq. 1. This score integrates the value $Q(s,a)$ estimated by a PRM, the prior probability $P(s,a)$ given by the LLM, and the visit count $N(s,a)$.

$$\begin{aligned}\arg\max_a \Bigg[& Q(s,a) + P(s,a)\frac{\sqrt{N(s,b)}}{1+N(s,a)} \\ & \cdot \left(c_1 + \log\left(\frac{N(s,b)+c_2+1}{c_2}\right)\right)\Bigg]\end{aligned} \tag{1}$$

where s denotes the current state, *i.e.*, the parent node, which includes the question and all previously generated reasoning steps. a represents a specific candidate action from state s, *i.e.*, a possible next reasoning step to be generated. $N(s, b)$ corresponds to the total number of visits to the parent node s. The hyperparameters c_1 and c_2 are used to control the influence of the policy term relative to the value term (exploration degree). In our experiments, we set c_1 = 1.25 and c_2 = 19,652.

2. **Conditional Self-refinement.** The winning candidate from the Best-of-N stage then undergoes a conditional self-refinement process. Specifically, self-refinement is triggered only if its PRM score falls below a predefined threshold. In such cases, a revised step is generated based on the self-feedback, and adopted only if its PRM score exceeds that of the original step; otherwise, the original step is retained.

3. **Monte Carlo Tree Search.** Finally, the optimal candidate step that passes both the Best-of-N and conditional self-refinement stages is officially selected as the final step and appended to the current search trajectory. It then becomes the new leaf node from which the subsequent search iteration will expand.

We conceptualize this pipeline as "Deep Scaling" applied to the simulation phase, which is the most critical component of the MCTS. In a standard MCTS, the simulation (or "rollout") typically starts from the current reasoning step and generates the remaining part of the solution with a default policy. Subsequently, an evaluation score based on the correctness of the final answer is back-propagated to update the value estimates of all nodes along the reasoning path. In contrast, "Deep Scaling" reshapes the above process. Since the PRM can provide immediate and precise value assessments for each intermediate step, it eliminates the need to complete the entire solution before evaluating node quality. As a result, we focus computational resources on the deep scaling of each intermediate step, performing a more deliberate exploration through multi-sampling and iterative refinement. In summary, the introduction of the step-level verifier provides accurate and efficient guidance for LLMs during test-time scaling. Meanwhile, the integration of multiple training-free TTS strategies ensures both the breadth and depth of exploration in the solution space, achieving a comprehensive performance improvement.

4 Experimental Setup

Models. For LLMs, we include Qwen2.5-3B-Instruct, Qwen2.5-7B-Instruct, Qwen2.5-14B-Instruct (Yang et al., 2024), LLaMA3.1-8B-Instruct (Grattafiori et al., 2024), and Gemma3-4B-IT (Team et al., 2025) to span a range of model scales. We deliberately exclude RL-enhanced LLMs because their implicit CoT reasoning prior to the final answer lacks a clear step-by-step structure. Moreover, we aim to explore how to effectively activate the inherent reasoning potential of instruction-tuned LLMs, and provide insights for training-based TTS methods. For PRMs, we select Qwen2.5-Math-PRM-7B (Zhang et al., 2025d) as the step-level verifier for process verification, for reasons detailed in Section 2.2.

Datasets. We mainly select three mathematical and reasoning datasets: MATH500 (Hendrycks et al., 2021), AIME24 (MAA, 2024), and GPQA Diamond (Rein et al., 2024). They span a range of domains including mathematical computation, complex reasoning, and scientific questions, and present a significant difficulty level.

Implementation. We implement the Hybrid Test-Time Scaling method based on OpenR[2], an open-source framework designed to integrate key components for enhancing the reasoning capabilities of LLMs. All experiments are run once on two 80G A800 GPUs based on vLLM (Kwon et al., 2023) framework for efficient LLM inference, with one GPU deploying LLM service and another for PRM service. All hyperparameters are consistent with the model's default generation configuration.

Evaluation. We evaluate the reasoning ability of LLMs with three metrics:

- **Maj@k** measures the majority vote correctness from k independently sampled outputs based on self-consistency (Wang et al., 2023b).

- **RM@k** measures the best solution correctness from k sampled outputs selected by the specific reward model. Our strategy is to select the reasoning path that exhibits the maximum of per-path minimum PRM scores as the final solution.

- **Pass@k** measures whether at least one of the model's k sampled outputs is correct.

[2] https://openreasoner.github.io

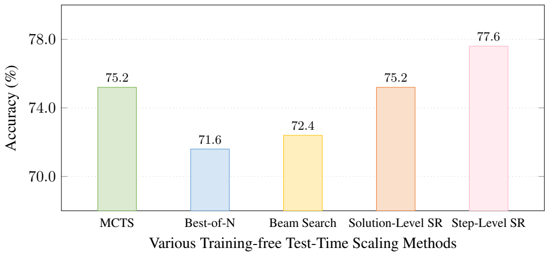

Figure 2: Comparisons between four common training-free TTS methods with Qwen2.5-7B-Instruct on the MATH500 dataset, where "SR" represents the basic self-refinement for one iteration.

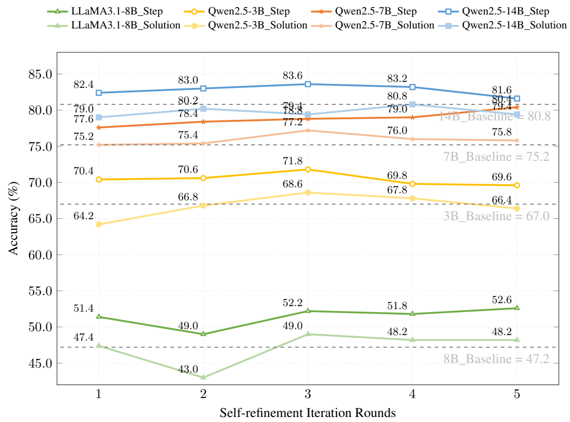

Figure 3: Multi-round iterative self-refinement at the solution-level and step-level on the MATH500 dataset.

Baselines. We compare the conditional step-level self-refinement method against other parallel scaling methods and solution-level self-refinement under identical verifier and iteration settings. As for hybrid test-time scaling, it incorporates MCTS, BoN, and step-level self-refinement; we thus regard pairwise combinations as comparative baselines. Additionally, we test solution-level hybrid scaling methods (OpenR) proposed by Wang et al. (2024) under equivalent sampling, search path, and refinement conditions.

5 Results and Analyses

5.1 Preliminary Experiments for Conditional Step-level Self-refinement

Basic step-level self-refinement outperforms other training-free test-time scaling methods. We first evaluated both solution-level and step-level self-refinement, each for one iteration, using the prompt templates detailed in Appendix A. As shown in Figure 2, our basic step-level self-refinement achieves superior reasoning performance compared to other training-free TTS baselines reported in Ding et al. (2025).

Sequential scaling at the step level is superior to solution level. Since self-refinement achieves sequential scaling through multi-round iterative optimization, we further increased the number of refinement iterations. We employed Qwen2.5-Math-PRM-7B as a verifier to conditionally accept a refined response only if it improved upon the original. Specifically, the solution quality was determined by the PRM score of its final answer. As presented in Figure 3, we observe that step-level self-refinement consistently outperforms solution-level across four different LLMs, revealing considerable advantages of fine-grained sequential scaling.

Reasonable conditions for improved step-level self-refinement. To further balance the performance and efficiency of step-level self-refinement, we employed Qwen2.5-7B-Instruct on the MATH500 dataset to investigate optimal refinement conditions. Based on prior knowledge, we considered the following four aspects:

- **Update Strategy.** "Cover" unconditionally replaces the current step with the refined step, while "PRM_Cover" conditionally updates only if the refined step's PRM score surpasses that of the previous iteration. As presented in Figure 4, with increasing iteration rounds, "PRM_Cover" leads to steady performance gains, whereas "Cover" causes serious performance degradation, even falling below the baseline. This indicates that unconditional reflection may prompt LLMs to incorrectly revise correct answers.

- **Number of Iterations.** Based on "PRM_Cover" strategy, we experimented with up to 10 iteration rounds to observe performance trends of sequential scaling. As depicted in Figure 5(a), reasoning performance peaked at the 5th round and plateaued thereafter. Therefore, we set a maximum of 5 iteration rounds as the condition.

The MATH500 dataset includes annotations for problem difficulty levels. We analyzed the sum of PRM scores for a solution corresponding to different problem difficulties and found a linear correlation between them, as illustrated in Figure 6 in Appendix D. In other words, we can predict problem difficulty with the help of PRM scores and thus determine whether it is necessary or worthwhile for LLMs to self-reflect. In this way, scaling is not performed for either overly easy or excessively

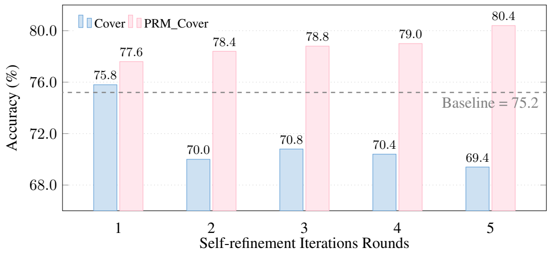

Figure 4: Experiments on different update strategies.

difficult problems, where the former can avoid overthinking, and the latter allows the model to "give up" strategically, thereby improving efficiency.

- **Necessity of Reflection.** To avoid hallucinations from reflecting on correct steps, we set a PRM threshold to skip reflection on high-confidence steps. After testing various PRM thresholds in Figure 5(b), we conservatively adopt 0.9 to flag correct steps that do not require self-reflection.

- **Worthiness of Reflection.** Given the complexity of reasoning tasks, LLMs may lack the capability to solve certain difficult problems. Therefore, we established a heuristic rule: if the PRM score improvement over N consecutive iteration rounds falls below a threshold, the step is deemed unlikely to be corrected and further refinement is abandoned to improve efficiency. Based on experiments shown in Figure 5(c), we adopted a condition of "PRM improvement over 2 rounds < 0.2" (gap@2 < 0.2) as the criterion for the LLM to give up refinement for intractable steps.

In summary, we validate the effectiveness of fine-grained sequential scaling and define the following specific conditions for improved step-level self-refinement used in our Hybrid Test-Time Scaling method: a maximum of 5 iteration rounds; results are overwritten only when PRM scores improve (PRM_Cover); and iteration stops if the PRM score exceeds 0.9 or if the PRM score improvement over two consecutive iteration rounds falls below 0.2.

5.2 Main Results for Hybrid Test-Time Scaling

Hybrid Test-Time Scaling generally improves the reasoning performance of instruction-tuned LLMs. To demonstrate the versatility of Hybrid Test-Time Scaling, we conducted main experiments on five instruction-tuned LLMs across 3B to 14B parameters from different families. As shown in Table 2, all configurations of Hybrid TTS are based

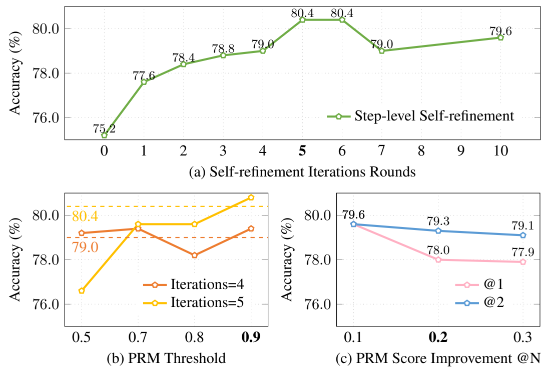

Figure 5: Conditions for improved step-level self-refinement with Qwen2.5-7B-Instruct on the MATH500 dataset, where @N represents between N successive iteration rounds.

on conditional step-level self-refinement with the number of samples N in BoN and the number of search paths k in tree-search as variables. Performance is evaluated using three metrics: Maj@k, RM@k, and Pass@k. Our method achieves significant performance gains over baselines across all models and datasets. Peak improvements reach 28.6% for LLaMA3.1-8B-Instruct on MATH500, 13.4% for Qwen2.5-14B-Instruct on AIME24, and 21.5% for Qwen2.5-3B-Instruct on GPQA Diamond. We also observe that Hybrid TTS effectively bridges the performance gap between model scales, enabling smaller LLMs to achieve the baseline reasoning performance of much larger ones. Notably, the performance of Qwen2.5-3B/7B-Instruct and Gemma3-4B-IT on the GPQA Diamond dataset even exceeds that of the RL-enhanced DeepSeek-R1-Distill-Qwen-7B (DeepSeek-AI et al., 2025), indicating that hybrid training-free TTS methods can effectively unlock the instruction-tuned LLM's latent reasoning potential.

Verification matters in smaller LLM test-time scaling. A comparison of performance metrics reveals a consistent pattern: Maj@k and RM@k scores are generally lower than the Pass@k score. This discrepancy is particularly pronounced in smaller-scale LLMs, reaching a gap of up to 20.7% for Gemma3-4B-IT on the GPQA Diamond dataset. This indicates that while smaller models possess strong latent reasoning capabilities (high Pass@k), their ability to consistently identify and select the best answer from a set of candidates is limited (lower Maj@k and RM@k). Therefore, to bridge this gap between latent potential and actualized performance, an accurate verification mechanism is vital—not only for final answer selection, but

Models	Best-of-1	Best-of-4			Best-of-8			Best-of-16		
	Baseline	`Maj@4`	`RM@4`	`Pass@4`	`Maj@8`	`RM@8`	`Pass@8`	`Maj@16`	`RM@16`	`Pass@16`
					MATH500					
Qwen2.5-3B	67.0	79.2	80.6	83.2	81.4	82.8	85.0	83.8	84.8	87.0
Qwen2.5-7B	75.2	84.0	84.8	85.4	84.2	85.4	86.6	85.6	86.2	88.2
Qwen2.5-14B	80.8	85.8	86.6	87.4	87.6	87.8	88.6	88.0	87.8	89.6
LLaMA3.1-8B	47.2	64.2	66.6	68.4	69.4	71.4	74.2	71.4	73.0	75.8
Gemma3-4B	75.6	80.4	80.6	82.4	82.4	82.6	84.2	85.0	84.0	87.0
					AIME24					
Qwen2.5-3B	6.7	10.0	16.7	16.7	16.7	13.3	16.7	16.7	16.7	16.7
Qwen2.5-7B	10.0	20.0	20.0	23.3	23.3	23.3	23.3	16.7	20.0	20.0
Qwen2.5-14B	13.3	23.3	26.7	26.7	23.3	26.7	30.0	30.0	26.7	36.7
LLaMA3.1-8B	10.0	10.0	10.0	10.0	10.0	10.0	16.7	16.7	13.3	20.0
Gemma3-4B	10.0	13.3	13.3	13.3	16.7	16.7	16.7	16.7	16.7	16.7
					GPQA Diamond					
Qwen2.5-3B	30.3	35.4	34.3	43.9	36.9	39.9	48.0	39.9	41.4	51.5
Qwen2.5-7B	36.4	41.4	40.9	45.5	45.5	43.9	51.0	44.4	43.9	51.0
Qwen2.5-14B	45.5	51.0	47.0	58.1	49.0	51.5	56.6	50.0	45.5	59.1
LLaMA3.1-8B	30.4	30.8	30.8	42.9	30.8	32.3	45.5	34.3	28.3	41.4
Gemma3-4B	30.8	33.3	29.8	39.9	37.9	35.4	47.5	37.9	32.8	53.5

Table 2: Main results of Hybrid Test-Time Scaling with instruction-tuned LLMs. Baseline represents their basic reasoning ability without test-time scaling, while other settings combine Best-of-N sampling and conditional step-level self-refinement in k-path tree search.

also for guiding the reasoning process toward the correct direction at each intermediate step.

High-quality verification plays a pivotal role in TTS performance. It is worth noting that a similar principle applies in reinforcement learning (RL), which also relies on a verifier (*e.g.*, reward model) to provide accurate feedback as supervision signals. To empirically validate the importance of verification quality, we present a comparative analysis using two PRMs of different capabilities: the state-of-the-art Qwen2.5-Math-PRM-7B and the weaker Math-Shepherd-Mistral-7B-PRM (Wang et al., 2023a). As shown in Table 3, while the performance guided by Math-Shepherd surpasses the baseline (75.2), it still lags significantly behind the results obtained with Qwen2.5-Math-PRM-7B. This outcome is consistent with our claim in Section 2.2 that an accurate verification is critical for maximizing reasoning performance.

The essence of Test-Time Scaling lies in the extensive exploration and precise exploitation of the LLM's solution space. This principle is demonstrated by observing how reasoning performance varies with an increasing number of samples and search paths. We find that all models exhibit a performance growth trend on the MATH500 dataset, suggesting that a comprehensive exploration of the solution space can enhance model reasoning capabilities. However, results on the GPQA dataset concurrently indicate that performance gains from such an expanded search space are unstable. For instance, when conducting multiple experiments under identical settings, the outcomes sometimes varied significantly, as shown in Figure 7 in Appendix E. We attribute this to the inherent instability of model generation at higher temperature settings. Nevertheless, their unexpectedly high accuracy scores (*e.g.*, 59.1 and 53.5) confirm that models can possess considerable reasoning potential even without continued training. This highlights the need for accurate external verification signals to enable more precise exploitation of the solution space, which is the core idea behind our Hybrid Test-Time Scaling-"*best-of-the-best*". Put another way, if we can further improve the accuracy of the verifier, it is highly plausible that more consistent performance improvements could be achieved solely through training-free methods.

5.3 Ablation Study

To validate the necessity of integrating all three components of our method, we conducted ablation studies on their pairwise combinations. The results across various models and datasets are summarized in Table 4. To ensure a fair comparison,

PRMs	Best-of-4			Best-of-8			Best-of-16		
	`Maj@4`	`RM@4`	`Pass@4`	`Maj@8`	`RM@8`	`Pass@8`	`Maj@16`	`RM@16`	`Pass@16`
Math-Shepherd-Mistral-7B-PRM	77.6	75.6	79.2	77.4	75.8	80.0	77.6	76.0	79.8
Qwen2.5-Math-PRM-7B	84.0	84.8	85.4	84.2	85.4	86.6	85.6	86.2	88.2

Table 3: Comparative experiments of Hybrid Test-Time Scaling with different PRMs using Qwen2.5-7B-Instruct on the MATH500 dataset.

Methods	Best-of-4			Best-of-8			Best-of-16		
	`Maj@4`	`RM@4`	`Pass@4`	`Maj@8`	`RM@8`	`Pass@8`	`Maj@16`	`RM@16`	`Pass@16`
Qwen2.5-3B + GPQA Diamond									
BoN+Self-Refinement	$36.4_{\texttt{Pass@1}}$			$37.9_{\texttt{Pass@1}}$			$40.4_{\texttt{Pass@1}}$		
MCTS+BoN	33.8	35.4	<u>40.9</u>	40.4	35.9	<u>44.4</u>	36.9	41.4	<u>49.0</u>
OpenR (Wang et al., 2024)	30.3	30.8	37.4	29.8	32.3	43.9	33.3	34.8	47.0
Hybrid Test-Time Scaling	35.4	34.3	**43.9**	36.9	39.9	**48.0**	39.9	41.4	**51.5**
LLaMA3.1-8B + MATH500									
BoN+Self-Refinement	$62.4_{\texttt{Pass@1}}$			$66.8_{\texttt{Pass@1}}$			$66.6_{\texttt{Pass@1}}$		
MCTS+BoN	62.2	63.6	<u>64.8</u>	65.2	68.8	<u>71.6</u>	69.6	70.6	<u>73.2</u>
OpenR (Wang et al., 2024)	48.6	47.6	54.2	60.2	60.2	63.0	60.4	60.0	61.2
Hybrid Test-Time Scaling	64.2	66.6	**68.4**	69.4	71.4	**74.2**	71.4	73.0	**75.8**
Qwen2.5-14B + AIME24									
BoN+Self-Refinement	$13.3_{\texttt{Pass@1}}$			$16.7_{\texttt{Pass@1}}$			$20.0_{\texttt{Pass@1}}$		
MCTS+BoN	13.3	16.7	16.7	16.7	23.3	<u>23.3</u>	20.0	23.3	<u>26.7</u>
OpenR (Wang et al., 2024)	16.7	20.0	<u>20.0</u>	16.7	16.7	16.7	23.3	23.3	23.3
Hybrid Test-Time Scaling	23.3	26.7	**26.7**	23.3	26.7	**30.0**	30.0	26.7	**36.7**

Table 4: Ablation study on pairwise combinations from our Hybrid Test-Time Scaling methods, where the best results are shown in **bold** and the second-best results are <u>underlined</u>. "BoN+Self-Refinement" represents one reasoning path for every question, so only `Pass@1` is reported. "MCTS+BoN" is parallel-only scaling, and "OpenR" is solution-level Hybrid Test-Time Scaling.

all tested combinations employ the same PRM verifier and operate at the step level. We also compare our method against OpenR (Wang et al., 2024), a solution-level baseline that combines MCTS with multi-round self-refinement under the identical refinement conditions, where the self-refinement serves as the MCTS simulation. The results indicate that while all pairwise combinations improve upon the baseline, none match the performance of our complete three-component hybrid strategy. This finding strongly suggests that each component contributes meaningfully to the overall performance, providing compelling support for our conclusion that a fine-grained hybrid inference paradigm yields superior performance.

A detailed analysis of the computational overhead for these methods is provided in Appendix C. In practice, a maximum of 5 iteration rounds and relatively conservative stopping conditions inevitably increases inference cost. However, in order to fully explore the upper bound of the reasoning capabilities of training-free TTS, we chose a strategy that maximizes reasoning performance.

6 Conclusion

In this paper, we propose Step-level Verifier-guided Hybrid Test-Time Scaling for LLM reasoning. First, we introduce a conditional step-level self-refinement method to balance performance and efficiency. Building on this, we synergistically combine this fine-grained sequential scaling method with representative parallel scaling techniques (*e.g.*, Best-of-N and MCTS) at the step level. The effectiveness of this hybrid inference paradigm demonstrates that different sequential and parallel scaling methods can be orthogonal and complementary. Finally, our large-scale experiments corroborate that the essence of test-time scaling is to deeply explore and precisely exploit the LLM's solution space. The success in enhancing smaller instruction-tuned LLMs via training-free TTS methods not only establishes the pivotal role of accurate step-level verifiers but also inspires a new direction for training-based TTS with smaller LLMs.

Limitations

We only perform inference once in the experiments, so there may be some performance fluctuations. However, the purpose of this paper is to explore the reasoning performance boundaries achievable by hybrid training-free TTS methods. This conclusion has already been confirmed by most experiments, and training-free TTS methods even have the potential to outperform training-based TTS methods.

Acknowledgments

This work was supported in part by the National Science Foundation of China (Nos. 62276056 and U24A20334), the Yunnan Fundamental Research Projects (No. 202401BC070021), the Yunnan Science and Technology Major Project (No. 202502AD080014), and the Program of Introducing Talents of Discipline to Universities, Plan 111 (No.B16009).

References

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and 1 others. 2024. Graph of thoughts: Solving elaborate problems with large language models. In *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, pages 17682–17690. AAAI Press.

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*, abs/2407.21787.

Kaiyan Chang, Songcheng Xu, Chenglong Wang, Yingfeng Luo, Xiaoqian Liu, Tong Xiao, and Jingbo Zhu. 2024. Efficient prompting methods for large language models: A survey. *ArXiv preprint*, abs/2404.01077.

Qiguang Chen, Libo Qin, Jinhao Liu, Dengyun Peng, Jiannan Guan, Peng Wang, Mengkang Hu, Yuhang Zhou, Te Gao, and Wanxiang Che. 2025. Towards reasoning era: A survey of long chain-of-thought for reasoning large language models. *ArXiv preprint*, abs/2503.09567.

Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qiuzhi Liu, Mengfei Zhou, Zhuosheng Zhang, Rui Wang, Zhaopeng Tu, Haitao Mi, and Dong Yu. 2024. Do not think that much for 2+3=? on the overthinking of o1-like llms. *ArXiv preprint*, abs/2412.21187.

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*, abs/2501.12948.

Yifu Ding, Wentao Jiang, Shunyu Liu, Yongcheng Jing, Jinyang Guo, Yingjie Wang, Jing Zhang, Zengmao Wang, Ziwei Liu, Bo Du, and 1 others. 2025. Dynamic parallel tree search for efficient llm reasoning. *ArXiv preprint*, abs/2502.16235.

Yuchun Fan, Yongyu Mu, Yilin Wang, Lei Huang, Junhao Ruan, Bei Li, Tong Xiao, Shujian Huang, Xiaocheng Feng, and Jingbo Zhu. 2025. Slam: Towards efficient multilingual reasoning via selective language alignment. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 9499–9515.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. 2023. Critic: Large language models can self-correct with tool-interactive critiquing. *ArXiv preprint*, abs/2305.11738.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *ArXiv preprint*, abs/2407.21783.

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. *ArXiv preprint*, abs/2103.03874.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2024. Large language models cannot self-correct reasoning yet. In *The Twelfth International Conference on Learning Representations*.

Yixin Ji, Juntao Li, Hai Ye, Kaixin Wu, Jia Xu, Linjian Mo, and Min Zhang. 2025. Test-time computing: from system-1 thinking to system-2 thinking. *ArXiv preprint*, abs/2501.02497.

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*, abs/2405.16265.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *ArXiv preprint*, abs/2001.08361.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Zhong-Zhi Li, Duzhen Zhang, Ming-Liang Zhang, Jiaxin Zhang, Zengyan Liu, Yuxuan Yao, Haotian Xu, Junhao Zheng, Pei-Jie Wang, Xiuyi Chen, and 1 others. 2025. From system 1 to system 2: A survey of reasoning large language models. *arXiv preprint arXiv:2502.17419*.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*.

Runze Liu, Junqi Gao, Jian Zhao, Kaiyan Zhang, Xiu Li, Biqing Qi, Wanli Ouyang, and Bowen Zhou. 2025. Can 1b llm surpass 405b llm? rethinking compute-optimal test-time scaling. In *Workshop on Reasoning and Planning for Large Language Models*.

MAA. 2024. American invitational mathematics examination. Online.

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 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Terufumi Morishita, Gaku Morio, Atsuki Yamaguchi, and Yasuhiro Sogawa. 2024. Enhancing reasoning capabilities of llms via principled synthetic logic corpus. *Advances in Neural Information Processing Systems*, 37:73572–73604.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. 2025. s1: Simple test-time scaling. *ArXiv preprint*, abs/2501.19393.

OpenAI. 2024. Openai o1 system card. Accessed: 2024-11-07.

Zhenting Qi, MA Mingyuan, Jiahang Xu, Li Lyna Zhang, Fan Yang, and Mao Yang. 2025. Mutual reasoning makes smaller llms stronger problem-solver. In *The Thirteenth International Conference on Learning Representations*.

Gollam Rabby, Farhana Keya, Parvez Zamil, and Sören Auer. 2024. Mc-nest–enhancing mathematical reasoning in large language models with a monte carlo nash equilibrium self-refine tree. *ArXiv preprint*, abs/2411.15645.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. 2024. Gpqa: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, and Arthur Guez. 2020. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–610.

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

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

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*, abs/2501.03124.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, and 1 others. 2025. Gemma 3 technical report. *ArXiv preprint*, abs/2503.19786.

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*, abs/2211.14275.

Chenglong Wang, Yang Gan, Yifu Huo, Yongyu Mu, Qiaozhi He, MuRun Yang, Bei Li, Tong Xiao, Chunliang Zhang, Tongran Liu, and 1 others. 2025. Gram: A generative foundation reward model for reward generalization. In *Forty-second International Conference on Machine Learning*.

Jun Wang, Meng Fang, Ziyu Wan, Muning Wen, Jiachen Zhu, Anjie Liu, Ziqin Gong, Yan Song, Lei Chen, Lionel M Ni, and 1 others. 2024. Openr: An open source framework for advanced reasoning with large language models. corr, abs/2410.09671, 2024b. doi: 10.48550. *ArXiv preprint*, abs/2410.09671.

Peiyi Wang, Lei Li, Zhihong Shao, RX Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. 2023a. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. *ArXiv preprint*, abs/2312.08935.

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

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. 2024a. Inference scaling laws: An empirical analysis of compute-optimal inference for problem-solving with language models. *ArXiv preprint*, abs/2408.00724.

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. 2024b. Scaling inference computation: Compute-optimal inference for problem-solving with language models. In *The 4th Workshop on Mathematical Reasoning and AI at NeurIPS*, volume 24.

Tong Xiao and Jingbo Zhu. 2025. Foundations of large language models. *arXiv preprint arXiv:2501.09223*.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, and 1 others. 2024. Qwen2. 5 technical report. *ArXiv preprint*, abs/2412.15115.

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 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, and 1 others. 2025. Dapo: An open-source llm reinforcement learning system at scale. *ArXiv preprint*, abs/2503.14476.

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

Di Zhang, Jianbo Wu, Jingdi Lei, Tong Che, Jiatong Li, Tong Xie, Xiaoshui Huang, Shufei Zhang, Marco Pavone, Yuqiang Li, and 1 others. 2025a. Llama-berry: Pairwise optimization for olympiad-level mathematical reasoning via o1-like monte carlo tree search. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 7315–7337.

Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. 2025b. Generative verifiers: Reward modeling as next-token prediction. In *The Thirteenth International Conference on Learning Representations*.

Qiyuan Zhang, Fuyuan Lyu, Zexu Sun, Lei Wang, Weixu Zhang, Zhihan Guo, Yufei Wang, Irwin King, Xue Liu, and Chen Ma. 2025c. What, how, where, and how well? a survey on test-time scaling in large language models. *ArXiv preprint*, abs/2503.24235.

Yunxiang Zhang, Muhammad Khalifa, Lajanugen Logeswaran, Jaekyeom Kim, Moontae Lee, Honglak Lee, and Lu Wang. 2024b. Small language models need strong verifiers to self-correct reasoning. *ArXiv preprint*, abs/2404.17140.

Zhenru Zhang, Chujie Zheng, Yangzhen Wu, Beichen Zhang, Runji Lin, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. 2025d. The lessons of developing process reward models in mathematical reasoning. *ArXiv preprint*, abs/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*, abs/2411.14405.

Tong Zheng, Lichang Chen, Simeng Han, R Thomas McCoy, and Heng Huang. 2025. Learning to reason via mixture-of-thought for logical reasoning. *ArXiv preprint*, abs/2505.15817.

A Meta-prompts for Self-refinement

Following the principles of efficient prompting outlined in Chang et al. (2024), we designed meta-prompts for both step- and solution-level self-refinement to achieve a performance-efficiency trade-off.

Prompt for step-level self-refinement

CoT Prompt: Please reason step by step, and put your final answer within boxed{}.

Critic Prompt: There is a weak reasoning step in the solution, please provide a strict reflection to correct only this one step with less than 150 tokens. Don't output the complete solution.

Rewrite Prompt: Please refine the weak answer according to your Reflection. Not allowed to use code to solve the question. Please reason step by step, and put your final answer within boxed{}.

Prompt for solution-level self-refinement

CoT Prompt: Please reason step by step, and put your final answer within boxed{}.

Critic Prompt: There is a weak solution, please provide a strict reflection to correct it.

Rewrite Prompt: Please refine the weak answer according to your Reflection. Not allowed to use code to solve the question. Please reason step by step, and put your final answer within boxed{}.

Prompt for step-level self-refinement (GPQA)

CoT Prompt: Please reason step by step, and put your final answer within boxed{LETTER}(without quotes) where LETTER is one of ABCD.

Critic Prompt: There is a weak reasoning step in the solution, please provide a strict reflection to correct only this one step with less than 150 tokens. Don't output the complete solution.

Rewrite Prompt: Please refine the weak answer according to your Reflection. Not allowed to use code to solve the question. Please reason step by step, and put your final answer within boxed{LETTER}(without quotes) where LETTER is one of ABCD.

Prompt for solution-level self-refinement (GPQA)

CoT Prompt: Please reason step by step, and put your final answer within boxed{LETTER}(without quotes) where LETTER is one of ABCD.

Critic Prompt: There is a weak solution, please provide a strict reflection to correct it.

Rewrite Prompt: Please refine the weak answer according to your Reflection. Not allowed to use code to solve the question. Please reason step by step, and put your final answer within boxed{LETTER}(without quotes) where LETTER is one of ABCD.

B The Algorithm for Hybrid Test-Time Scaling

We combine our proposed conditional step-level self-refinement with two classical parallel scaling methods, Best-of-N and Tree Search, under the guidance of a step-level verifier, and design an algorithm to describe the hybrid strategy.

Algorithm 1 Hybrid Test-Time Scaling

Input: Problem P, search paths N_p, samples N_s, self-refinement iterations N_r, LLM, PRM
Output: Final Solution s
Initialize root node n_0 from P
for $i = 1$ to N_p **do** ▷ **Tree Search**
  Initialize $n_{current} \leftarrow n_0$, $p_i \leftarrow$ ""
  while <eos> is not in $n_{current}.text$ **do**
    Generate N_s candidate steps $\mathcal{S}$ using LLM(P, p_i) ▷ Sampling
    Evaluate scores $\mathcal{V}$ for steps in $\mathcal{S}$ using PRM$(P, p_i, \cdot)$ to calculate P-UCB ▷ Evaluation
    Select s_{best} from $\mathcal{S}$ based on P-UCB as next step $n_{next}.text$ ▷ **Best-of-N**
    for $j = 1$ to N_r **do**
      if IsRefinement($n_{next}.text$) **then** ▷ **Conditional Self-refinement**
        $C \leftarrow$ LLM(`CritiquePrompt`$(P, p_i, n_{next}.text)$) ▷ Reflection
        $r \leftarrow$ LLM(`RefinePrompt`$(P, p_i, n_{next}.text, C)$) ▷ Revision
        if PRM$(P, p_i, r) >$ PRM$(P, p_i, n_{next}.text)$ **then**
          $n_{next}.text \leftarrow r$
        end if
      end if
    end for
    $p_i \leftarrow$ Concatenate$(p_i, n_{next}.text)$
  end while
  $\mathcal{P} \leftarrow \mathcal{P} \cup \{p_i\}$
end for
$s \leftarrow$ Majority Voting / Reward Model Scoring$(\mathcal{P})$
return s.

Methods	Best-of-4			Best-of-8			Best-of-16		
	FLOPs	# Tokens	Acc.	FLOPs	# Tokens	Acc.	FLOPs	# Tokens	Acc.
Qwen2.5-3B + GPQA Diamond									
BoN+Self-Refinement	7.41×10^9	9580.23	36.4	8.12×10^9	14422.07	37.9	**9.33**$\times10^9$	**22583.35**	**40.4**
MCTS+BoN	7.37×10^9	9276.94	40.9	**10.34**$\times10^9$	**29424.34**	**44.4**	17.80×10^9	80049.37	49.0
Hybrid Test-Time Scaling	**9.69**$\times10^9$	**25006.00**	**43.9**	14.25×10^9	55957.14	48.0	22.62×10^9	112728.73	51.5
Qwen2.5-14B + AIME24									
BoN+Self-Refinement	34.72×10^9	16418.77	13.3	36.29×10^9	20230.50	16.7	**51.83**$\times10^9$	**58182.17**	**20.0**
MCTS+BoN	33.21×10^9	12723.53	16.7	**47.95**$\times10^9$	**48700.07**	**23.3**	91.48×10^9	154984.13	26.7
Hybrid Test-Time Scaling	**50.20**$\times10^9$	**54189.77**	**26.7**	122.79×10^9	231410.97	30.0	150.21×10^9	298355.30	36.7

Table 5: The computational overhead of different combinations of training-free TTS methods.

C The Computational Overhead of Different Combinations of TTS Methods

As stated in Wu et al. (2024a), the inference FLOPs can be estimated following the Eq.2 defined in Kaplan et al. (2020):

$$\text{FLOPs}(N, T; S) \approx 2N + 2n_{\text{layer}} \times T \times d_{\text{model}} \quad (2)$$

where FLOPs depends on the number of model parameters N, the number of generated tokens T and the model architecture settings n_{layer} and d_{model}.

To fairly compare the cost-effectiveness across parallel, sequential, and hybrid methods under equal compute budgets, we consider fixed model configurations. In this setting, the inference FLOPs are primarily determined by the total number of generated tokens, which is aggregated from all generative steps within our framework, primarily from:

- Best-of-N (BoN): Sampling multiple step candidates based on the question and context.
- Tree Search (MCTS): Expanding nodes (steps) based on the question and context.
- Self-refinement: Generating an initial stepwise response based on the question and context, reflecting on potentially erroneous steps, and rewriting them based on the feedback.

Table 5 presents the average Inference FLOPs per question, average generation length, and performance for Qwen2.5-3B/14B-Instruct on the GPQA Diamond and AIME24 benchmarks, respectively. From these results, we observe that under comparable computational costs (*e.g.*, Best-of-16 for BoN+Self-Refinement, Best-of-8 for MCTS+BoN, and Best-of-4 for Hybrid TTS), Hybrid Test-Time Scaling consistently achieves superior performance. In contrast, using only pairwise combinations of these strategies (*e.g.*, MCTS+BoN or BoN+Self-Refinement)—even with a larger sampling budget—leads to more limited performance gains. Only the synergistic combination of all three strategies allows the model to fully realize its potential with training-free TTS. This demonstrates the cost-effectiveness advantage of the Hybrid Test-Time Scaling strategy.

D Verifier-indicated Difficulty Levels

We apply the "PRM_Cover" update strategy with Qwen2.5-7B-Instruct for five iterations on the MATH500 dataset. In each iteration, we calculate the sum of PRM scores for each solution corresponding to a problem. Then, we aggregate and average the PRM scores for problems of the same difficulty level, using this average solution-level PRM score to represent the difficulty of that category of problems. As shown in Figure 6, we found a linear correlation between the solution-level PRM and the problem difficulty level.

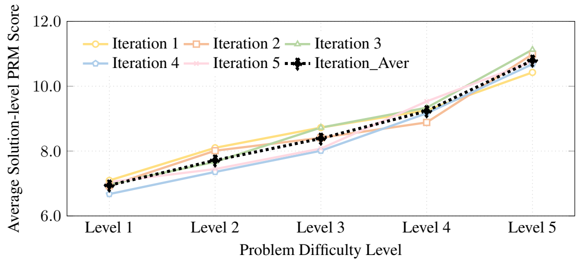

Figure 6: The correlation between solution-level PRM and problem difficulty level, where the black dashed line represents the average of all iterations.

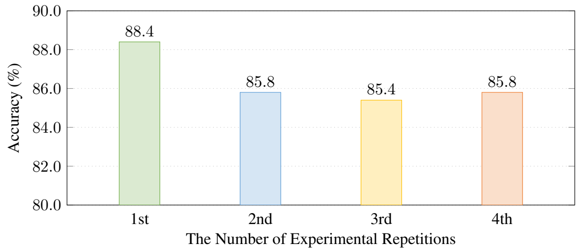

Figure 7: Results of repeated trials under exactly the same experimental setup.

E Instability of LLM during Training-free Test-Time Scaling

In our experiments, we typically perform inference only once for a set of setting. However, during the experimental process, we found that the model's performance can vary significantly under the same setting. Figure 7 shows the results of the Qwen2.5-7B-Instruct on the MATH500 dataset with the setting of Best-of-4 and 16-path search.

F Case Study of Step-level and Solution-level Self-refinement

Qwen2.5-7B-Instruct performed 5 rounds of step- and solution-level self-refinement on its initial generation for the same problem. The solution-level self-refinement failed to correct the error through iteration, whereas step-level self-refinement corrected the error in **Step3** after the first round, with its PRM score improving from 0.0596 to 0.8711.

Case Study

Question: The binary number 10101001110_2 is equal to what number in base eight?

Ground Truth: Since $2^3 = 8$, we may convert between base 2 and base 8 representations by replacing each block of three digits in base 2 with its equivalent in base 8. In this case, we begin by noticing that the last three digits are worth $110_2 = 6_8$. The next block of three digits is $001_2 = 1_8$. Continuing, we find that the next two digits (moving right-to-left) are $101_2 = 5_8$ and $010_2 = 2_8$. Altogether, we find that $10101001110_2 = \boxed{2516_8}$.

Initial Response: To convert the binary number 10101001110_2 to an octal (base eight) number, we can follow these steps:

1. **Group the binary digits into sets of three, starting from the right:** $10101001110_2 \rightarrow 101\,010\,011\,110$
2. **Convert each group of three binary digits to its octal equivalent:**
   - $101_2 = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 4 + 0 + 1 = 5_8$
   - $010_2 = 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 0 + 2 + 0 = 2_8$
   - $011_2 = 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 0 + 2 + 1 = 3_8$
   - $110_2 = 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 4 + 2 + 0 = 6_8$
3. **Combine the octal digits:** $10101001110_2 = 5236_8$
   Thus, the binary number 10101001110_2 is equal to $\boxed{5236_8}$. (Incorrect)

Solution-Level: Let's convert the binary number 10101001110_2 to octal equivalent step by step.

1. **Group the binary digits into sets of three, starting from the right:** $10101001110_2 \rightarrow 101\,010\,011\,110$
2. **Convert each group of three binary digits to its octal equivalent:**
   - For 101_2: $1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 4 + 0 + 1 = 5_8$
   - For 010_2: $0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 0 + 2 + 0 = 2_8$
   - For 011_2: $0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 0 + 2 + 1 = 3_8$
   - For 110_2: $1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 4 + 2 + 0 = 6_8$
3. **Combine the octal digits:** $10101001110_2 = 5236_8$

Thus, the binary number 10101001110_2 is equal to $\boxed{5236_8}$. (Incorrect)

Initial Step3: $10101001110_2 \rightarrow 101\,010\,011\,110$

Refined Step3: $10101001110_2 \rightarrow 010\,101\,001\,110$

Step-Level: Let's convert the binary number 10101001110_2 to base eight step by step.

1. **Group the binary digits into sets of three, starting from the right:** $10101001110_2 \rightarrow 010\,101\,001\,110$. We added a leading zero to the leftmost group to make it a set of three digits.
2. **Convert each group of three binary digits to its equivalent octal digit:**
   - $010_2 = 2_8$
   - $101_2 = 5_8$
   - $001_2 = 1_8$
   - $110_2 = 6_8$
3. **Combine the octal digits to form the final octal number:** $010\,101\,001\,110_2 \rightarrow 2516_8$

Therefore, the binary number 10101001110_2 is equal to $\boxed{2516_8}$. (Correct)