Every Rollout Counts: Optimal Resource Allocation for Efficient Test-Time Scaling

Xinglin Wang , Yiwei Li , Shaoxiong Feng , Peiwen Yuan , Yueqi Zhang , Jiayi Shi , Chuyi Tan , Boyuan Pan , Yao Hu , Kan Li *

Posted Date: 10 June 2025

doi: 10.20944/preprints202506.0780.v1

Keywords: Test-Time Scaling; resource allocation; large language model

 doi:10.20944/preprints202506.0780.v1

Article

Every Rollout Counts: Optimal Resource Allocation for Efficient Test-Time Scaling

Xinglin Wang [1], Yiwei Li [1], Shaoxiong Feng [2], Peiwen Yuan [1], Yueqi Zhang [1], Jiayi Shi [1], Chuyi Tan [1], Boyuan Pan [2], Yao Hu [2] and Kan Li [1,*]

[1] School of Computer Science, Beijing Institute of Technology
[2] Xiaohongshu Inc
* Correspondnece: likan@bit.edu.cn

Abstract: Test-Time Scaling (TTS) improves the performance of Large Language Models (LLMs) by using additional inference-time computation to explore multiple reasoning paths through search. Yet how to allocate a fixed rollout budget most effectively during search remains underexplored, often resulting in inefficient use of compute at test time. To bridge this gap, we formulate test-time search as a resource allocation problem and derive the optimal allocation strategy that maximizes the probability of obtaining a correct solution under a fixed rollout budget. Within this formulation, we reveal a core limitation of existing search methods: solution-level allocation tends to favor reasoning directions with more candidates, leading to theoretically suboptimal and inefficient use of compute. To address this, we propose Direction-Oriented Resource Allocation (DORA), a provably optimal method that mitigates this bias by decoupling direction quality from candidate count and allocating resources at the direction level. To demonstrate DORA's effectiveness, we conduct extensive experiments on challenging mathematical reasoning benchmarks including MATH500, AIME2024, and AIME2025. The empirical results show that DORA consistently outperforms strong baselines with comparable computational cost, achieving state-of-the-art accuracy. We hope our findings contribute to a broader understanding of optimal TTS for LLMs.

Keywords: Test-Time Scaling; resource allocation; large language model

1. Introduction

As the challenges of scaling up computation and data resources for pretraining continue to grow, scaling test-time computation has emerged as a critical paradigm for enhancing model performance [1–3]. By allocating additional computation at inference time, Test-Time Scaling (TTS) improves the performance of LLMs on complex tasks such as mathematical reasoning by enabling deeper exploration of possible solutions [4–6]. One prominent approach to scaling test-time computation is through search, where diverse candidate solutions are proposed and filtered using a Process Reward Model (PRM) to guide the procedure [2,3,7–9]. By pruning low-quality paths early and focusing computation on more promising ones, these strategies help steer the search process toward trajectories that are more likely to yield correct answers [10].

While these strategies yield promising performance gains, the question of how to optimally allocate a fixed rollout budget across competing candidate trajectories remains underexplored. In practice, existing strategies rely on human-designed heuristics (Figure 1): preserving certain number of high-quality candidates (Beam Search) [2], promoting diversity (DVTS) [8], or balancing exploration and exploitation (REBASE) [3]. While these intuitions offer practical value, they lack a principled foundation and do not provide guarantees of optimality, such as maximizing the probability of obtaining a correct solution. As a result, rollout budgets may be allocated inefficiently, limiting the effectiveness of test-time computation.

 doi:10.20944/preprints202506.0780.v1

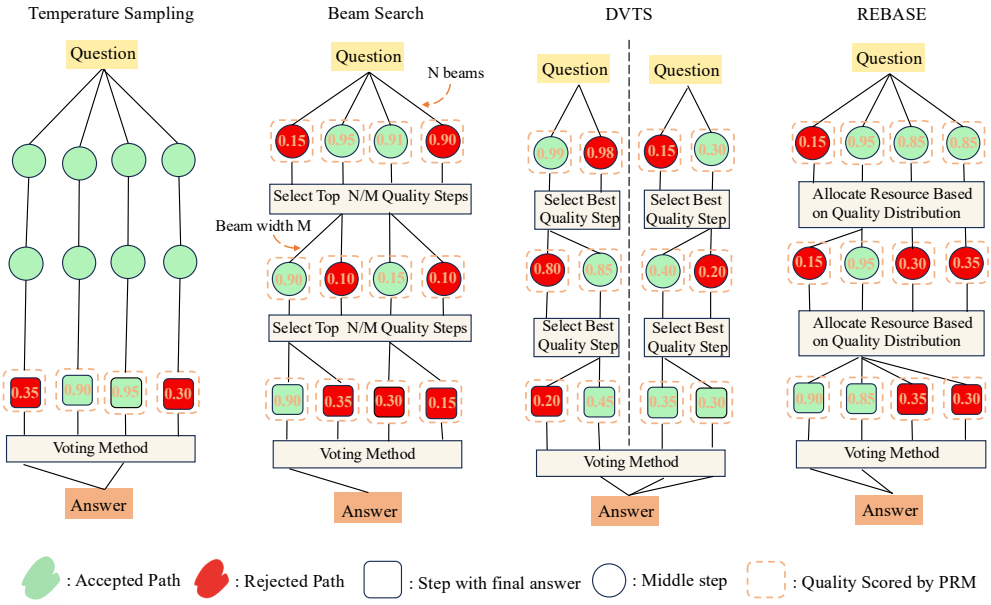

Figure 1. Comparison of different parallel Test-Time search strategies.

To bridge this gap, we formulate test-time search as a resource allocation problem, where the goal is to maximize the probability of obtaining a correct solution under a fixed rollout budget (Section 3.1). Based on this formulation, we derive the theoretical form of the optimal allocation strategy and revisit existing search methods through a unified lens. We show that, under the assumption that candidate solutions are independent, several widely used strategies approximate the optimal allocation corresponding to different assumptions about the reliability of the reward estimates. However, this independence assumption does not hold in practice, as many candidates share the same underlying reasoning direction [11,12]. Our theoretical analysis further shows that solution-level allocation is suboptimal: it conflates direction quality with candidate count, biasing the allocation toward overrepresented directions and leading to inefficient use of test-time compute (Section 3.2).

To address this issue, we propose Direction-Oriented Resource Allocation (DORA), a provably optimal method that corrects for this allocating bias by decoupling direction quality from candidate count and allocating resources at the direction level. To validate the effectiveness of DORA, we evaluate it on the challenging mathematical benchmarks MATH500 [13], AIME2024 [14], and AIME2025 across a broad range of rollout budgets and policy models. The empirical results show that DORA consistently outperforms strong baseline strategies under comparable computational budgets, highlighting its ability to improve the effectiveness of each rollout and enhance the overall efficiency of TTS.

2. Setup & Preliminaries

2.1. Problem Formulation

We formulate the parallel search process under a unified framework, defined by the tuple (π, Q, O, V, N), where $\pi(a \mid \tau)$ is a policy model that generates an action a (reasoning step) given a partial solution $\tau = (x, a_1, \ldots, a_i)$, where x denotes the input problem; $Q : \tau \mapsto [0, 1]$ is the Process Reward Model (PRM), which scores the quality of a partial or complete solution; $O : \mathbb{R}^N \to \mathbb{N}_+^N$ is the resource allocation strategy, dynamically assigning computational budget based on solution scores; V is the voting method that aggregates final answers from completed solutions to select the most likely correct final answer (e.g., via majority voting, best-of-N, or weighted best-of-N); and N is the total rollout budget of parallel explorations.

The parallel search process can be summarized as Algorithm A1. Specifically, the process iteratively expands a set of partial solutions using the policy π, collects complete solutions, and redistributes the rollout budget via the allocation strategy O based on intermediate rewards from Q. Once sufficient complete solutions are gathered, the final answer is selected using the voting method V.

 doi:10.20944/preprints202506.0780.v1

2.2. Parallel Search Method

We consider four parallel TTS methods which are popularly used in practice: Temperature Sampling [1], Beam Search [2], Diverse Verifier Tree Search (DVTS) [8], and Reward Balanced Search (REBASE) [3]. As pointed out by Snell et al. [2], lookahead search is inefficient due to sequential sampling, so we do not include it or other methods involving lookahead operations, such as Monte Carlo Tree Search (MCTS).

Based on the unified framework above, we now analyze these strategies from the perspective of resource allocation. While sharing the same overall structure, they differ solely in their choice of allocation function $O(\boldsymbol{R})$, which determines how the total rollout budget N is distributed across candidate solutions based on their PRM scores. We denote the number of rollouts assigned to the i-th candidate τ_j as $O(\boldsymbol{R})_i$, where O is the allocation function and $\boldsymbol{R} = \{R_1, \ldots, R_k\}$ is the vector of PRM scores.

Temperature Sampling.

This method performs sampling purely from the policy model, without using reward information for rollout allocation. All candidates are treated equally, and each receives one rollout. External reward signals may still be used at the final answer selection stage, e.g., through best-of-N or weighted best-of-N voting.

$$O_{\text{Temp}}(\boldsymbol{R})_i = 1. \tag{1}$$

Beam Search.

Beam Search selects the top $K = N/M$ candidates based on their PRM scores, where M is the number of rollouts assigned per candidate (i.e., the beam width). Only the top-K receive any rollout allocation, while the rest are discarded:

$$O_{\text{Beam}}(\boldsymbol{R})_i = \begin{cases} M, & \text{if } i \in \text{Top-}K(\boldsymbol{R}), \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

DVTS.

To encourage exploration across diverse solution branches, DVTS partitions the k candidates into $K = N/M$ disjoint groups of size M, corresponding to independent subtrees. Within each group, it performs a local Beam Search by selecting the candidate with the highest PRM score and assigning it M rollouts. Only one candidate per group receives any resource, and groups do not share information:

$$O_{\text{DVTS}}(\boldsymbol{R})_i = \begin{cases} M, & \text{if } i = \arg\max_{j \in \mathcal{G}(i)} R_j, \\ 0, & \text{otherwise,} \end{cases} \tag{3}$$

where $\mathcal{G}(i)$ denotes the group containing candidate i.

REBASE.

Instead of selecting a fixed number of candidates, REBASE distributes the total rollout budget more smoothly based on the relative quality of each candidate to balance exploitation and exploration. It applies a softmax over the PRM scores R_i to compute allocation weights, and assigns rollouts proportionally:

$$O_{\text{REBASE}}(\boldsymbol{R})_i = \text{round}(N \cdot w_i), \quad \text{where } w_i = \frac{e^{R_i/T_b}}{\sum_j e^{R_j/T_b}}. \tag{4}$$

where T_b is a temperature parameter controlling the sharpness of the allocation.

 doi:10.20944/preprints202506.0780.v1

3. Optimal Parallel Search for Test-Time Scaling

While previous parallel search methods enable efficient TTS by exploring multiple reasoning paths simultaneously, their effectiveness critically depends on how the fixed compute budget (i.e., number of rollouts) is allocated across candidate solutions. We focus on the following question:

> *Given a fixed rollout budget, how should one allocate resources across candidate reasoning paths to maximize performance (i.e., the success rate of achieving a correct solution)?*

We are the first to formulate this problem and study the associated parallel search strategies, setting our work apart from previous parallel search studies [3,8,15]. To address this, we introduce a Bayesian probabilistic model of solution correctness, and derive an allocation strategy that maximizes expected success under a rollout budget constraint.

3.1. Theoretical Formulation of Optimal Resource Allocation

We aim to allocate a fixed rollout budget N across k candidate reasoning paths to maximize the probability of solving the problem correctly, i.e., obtaining at least one successful solution. Let $p_i \in [0,1]$ denote the (unknown) success probability of the i-th candidate τ_i when sampled once.

Assumption 1. *The success events of different candidate solutions are independent.*

Under Assumption 1, the probability of obtaining at least one success under an allocation vector $\boldsymbol{B} = \{B_i\}_{i=1}^k$ is given by:

$$\mathbb{P}(\text{success}) = 1 - \prod_{i=1}^{k} (1 - p_i)^{B_i}. \tag{5}$$

Since the true values of p_i are unknown, we adopt a Bayesian modeling approach to capture the uncertainty in their estimation. In practice, p_i is often approximated using the Process Reward Model (PRM) score $R_i = Q(\tau_i)$ [16–20], which serves as a proxy for the probability of correctness. However, these estimates are subject to considerable noise due to imperfections in the policy model, variations in decoding temperature, and inherent sampling randomness. To model this uncertainty explicitly, we treat each p_i as a latent variable and place a Beta prior over it. Specifically, we normalize the PRM score into $w_i \in (0,1)$, and define:

$$p_i \sim \text{Beta}(\kappa w_i, \kappa(1 - w_i)), \tag{6}$$

where $\kappa > 0$ controls the concentration of the prior around its mean. Larger values of κ correspond to higher confidence in the PRM estimate w_i, while smaller values encode greater uncertainty (see Appendix C for more details).

Our goal is to maximize the probability of obtaining at least one successful solution. Under the Bayesian model, this is equivalent to minimizing the expected joint failure:

$$\min_{\sum B_i = N} \mathbb{E}\left[\prod_{i=1}^{k} (1 - p_i)^{B_i}\right]. \tag{7}$$

This defines a convex optimization problem over the rollout allocation vector $\boldsymbol{B} = \{B_i\}_{i=1}^k$. By applying the Karush-Kuhn-Tucker (KKT) conditions, we characterize the limiting behavior of the optimal allocation (see Appendix B.1 for details of proof):

Proposition 1 (Limiting Behavior of Optimal Allocation)**.** *Let $O^\star(\boldsymbol{w})_i$ denote the optimal rollout allocation for candidate i, where $\boldsymbol{w} = \{w_1, \ldots, w_k\}$ are the normalized PRM scores. Then:*

 doi:10.20944/preprints202506.0780.v1

- *When $\kappa \to 0$, the optimal allocation assigns one rollout to each of the top-$\min(k, N)$ candidates with highest w_i scores:*

$$O^\star(\boldsymbol{w})_i = \begin{cases} 1, & \text{if } i \in \text{Top-}\min(k, N) \text{ of } \boldsymbol{w}, \\ 0, & \text{otherwise}, \end{cases}$$

  with the remaining $N - \min(k, N)$ rollouts arbitrarily assigned.
- *When $\kappa \to \infty$, the optimal allocation converges to a deterministic allocation that assigns all rollouts to the highest-scoring candidate:*

$$O^\star(\boldsymbol{w})_i = \begin{cases} N, & \text{if } i = \arg\max_j w_j, \\ 0, & \text{otherwise}. \end{cases}$$

- *When κ is fixed and finite, the optimal allocation approximately follows a shifted linear rule:*

$$O^\star(\boldsymbol{w})_i \approx (N + k\kappa) \cdot w_i - \kappa.$$

Proposition 1 shows that the optimal allocation strategy evolves continuously with the confidence parameter κ. When $\kappa \to \infty$, the Beta prior becomes highly concentrated around the PRM estimate w_i, reflecting strong confidence in its accuracy. In this case, the optimal solution assigns the entire rollout budget to the top-ranked candidate, effectively recovering Beam Search with beam width $M = N$ (Equation 2) and fully exploiting the highest-scoring path.

Conversely, when $\kappa \to 0$, the Beta prior becomes maximally uncertain, collapsing to a Bernoulli mixture where each candidate has a binary chance of being correct or incorrect, with prior weight w_i. In this setting, relying heavily on any single PRM estimate becomes risky, as the scores provide no meaningful guidance. To mitigate this risk, the optimal strategy spreads the rollout budget across multiple candidates in proportion to their prior likelihoods. This reduces to sampling top candidates according to a multinomial distribution over w_i, a behavior closely aligned with temperature sampling used in stochastic decoding.

When κ is fixed and finite, the optimal allocation takes a smoothed, uncertainty-aware form that interpolates between the two extremes above. Specifically, the rollout budget is approximately distributed according to a shifted linear rule (Proposition 1), which closely matches the REBASE strategy (Equation 4). In this regime, the PRM scores are treated as informative but noisy, and the allocation strategy balances exploration and exploitation accordingly.
In practice, due to sampling noise and imperfections in the policy model, PRM scores carry considerable uncertainty. Consistent with this observation, we find that REBASE, which allocates rollouts in proportion to PRM scores, outperforms alternative strategies across a wide range of tasks (see Figure 3). This supports the relevance of the $\kappa \to 0$ setting, which we adopt as the default throughout the paper. Accordingly, we treat REBASE as the baseline solution-level allocation strategy in all subsequent analysis.

3.2. Suboptimality of Solution-Level Allocation

While REBASE is optimal under the assumption of candidate independence (Assumption 1), this condition often does not hold in practice. In particular, many candidate solutions share the same underlying reasoning direction [11,12], forming clusters of highly correlated outputs. The solution-level nature of REBASE leads to skewed allocation when candidate counts are imbalanced across reasoning directions.

To formalize this issue, we group candidate solutions into g reasoning directions. Let direction j contain k_j candidates, all sharing the same PRM score R_j, and let $\mathcal{E}_j$ denote the index set of these candidates.

 doi:10.20944/preprints202506.0780.v1

Under REBASE, rollout allocation is performed at the solution level, which implicitly induces a direction-level allocation according to Eq. 4:

$$B_j^{(\text{solution})} = \sum_{i \in \mathcal{E}_j} N \cdot \frac{e^{R_j}}{\sum_{l=1}^{g} k_l e^{R_l}} = N \cdot \frac{k_j e^{R_j}}{\sum_{l=1}^{g} k_l e^{R_l}}. \tag{8}$$

In contrast, the optimal allocation strategy would treat each reasoning direction as a single unit and assign rollouts in proportion to the softmax over direction-level scores:

$$B_j^{(\text{direction})} = N \cdot \frac{e^{R_j}}{\sum_{l=1}^{g} e^{R_l}}. \tag{9}$$

By comparing the induced solution-level allocation in Eq. 8 with the optimal direction-level allocation in Eq. 9, we derive the following proposition (see Appendix B.2 for details of proof):

Proposition 2 (Suboptimality of Solution-Level Allocation)**.** *When Assumption 1 does not hold, the solution-level allocation $B_j^{(solution)}$ is suboptimal: it does not match the optimal direction-level allocation $B_j^{(direction)}$ unless all directions contain the same number of candidate solutions, i.e., $k_j = k$ for all j.*

This result reveals a fundamental limitation of solution-level allocation: it implicitly favors reasoning directions with more candidate solutions (Figure 2). This bias results in inefficient use of the rollout budget, motivating our proposed method: Direction-Oriented Resource Allocation (DORA).

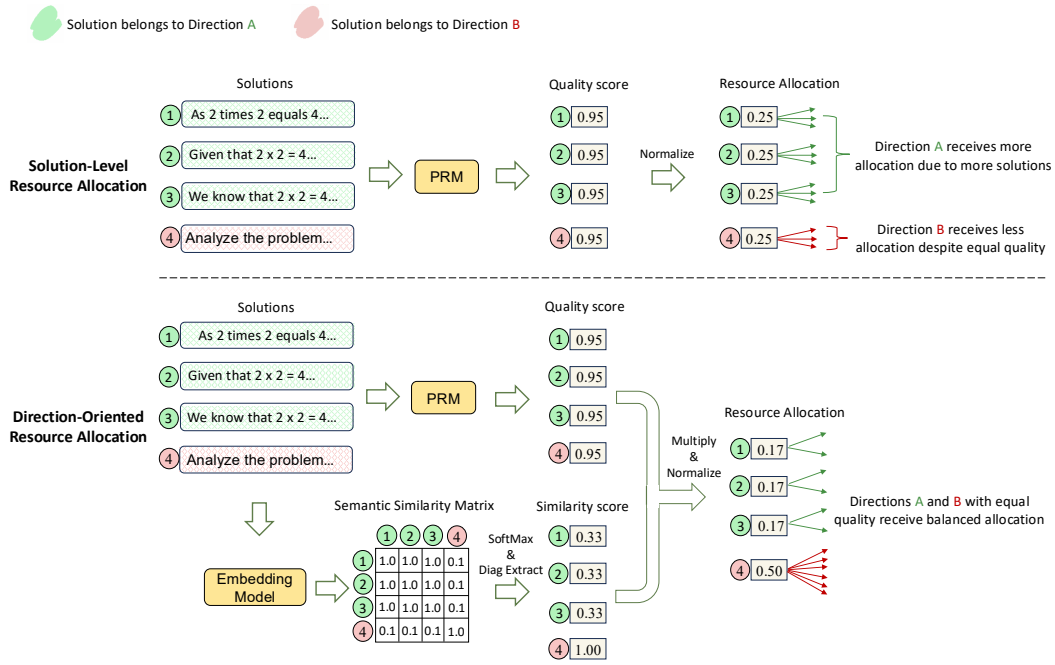

Figure 2. Comparison between Solution-Level Resource Allocation and proposed Direction-Oriented Resource Allocation (DORA).

3.3. Direction-Oriented Resource Allocation (DORA)

To address the bias introduced by solution-level allocation, we propose DORA, a method that adjusts rollout allocation by identifying and correcting for structural redundancy among candidate solutions. As illustrated in Figure 2, DORA incorporates semantic structure into the allocation process by softly clustering solutions into shared reasoning directions and assigning rollouts proportionally at the direction level, rather than treating each solution independently.

 doi:10.20944/preprints202506.0780.v1

Given a set of candidate solutions $\{\tau_1, \ldots, \tau_k\}$, DORA first estimates which solutions share reasoning structure by computing semantic embeddings $\boldsymbol{e}_i \in \mathbb{R}^d$ via a pretrained embedding model. These embeddings are used to construct a cosine similarity matrix $S \in \mathbb{R}^{k \times k}$:

$$S_{ij} = \frac{\boldsymbol{e}_i^\top \boldsymbol{e}_j}{\|\boldsymbol{e}_i\| \cdot \|\boldsymbol{e}_j\|}. \tag{10}$$

To avoid hard clustering and retain flexibility, we interpret the similarity between candidates as a soft assignment over directions. Specifically, we apply a row-wise softmax over S with temperature T_s, yielding an affinity matrix $P \in \mathbb{R}^{k \times k}$:

$$P_{ij} = \frac{e^{S_{ij}/T_s}}{\sum_{j'=1}^{k} e^{S_{ij'}/T_s}}. \tag{11}$$

The diagonal entry $\gamma_i = P_{ii}$ then measures the *semantic uniqueness* of solution τ_i, serving as a proxy for the inverse size of the solution's underlying direction.

Following the REBASE formulation in Eq. 4, we compute normalized quality weights w_i from PRM scores $R_i = Q(\tau_i)$ using a softmax with temperature T_b.

To incorporate semantic structure, we reweight each w_i by its uniqueness:

$$w_i' = \frac{w_i \cdot \gamma_i}{\sum_{j=1}^{k} w_j \cdot \gamma_j}. \tag{12}$$

This downweights redundant solutions and redistributes resources toward distinct reasoning directions.

Finally, rollouts are allocated proportionally:

$$B_i = \text{round}(N \cdot w_i'). \tag{13}$$

DORA balances rollouts across semantically distinct reasoning directions, mitigating the redundancy bias of solution-level methods like REBASE. As summarized in Theorem 1, DORA yields the optimal direction-level allocation under mild assumptions (See Appendix B.3 for the full derivation).

Theorem 1 (Optimality of DORA)**.** *Assume candidate solutions are grouped into g reasoning directions, where direction j consists of candidates indexed by $\mathcal{E}_j$, and all candidates in $\mathcal{E}_j$ share the same PRM score R_j. Then DORA recovers the optimal direction-level rollout allocation specified in Eq. 9.*

4. Experiments

4.1. Experimental Setup

We use Qwen2.5-Math-PRM-7B [21] as our Process Reward Model (PRM) due to its superior reward estimation performance [22,23]. For the policy models, we include Llama-3.2-1B-Instruct, Llama-3.2-3B-Instruct [24], and Qwen2.5-1.5B-Instruct [25], covering a range of model scales and architectures. Considering that existing open-source PRMs are primarily trained on mathematical tasks, we focus our evaluation on three challenging mathematical reasoning benchmarks: MATH500, AIME2024, and AIME2025. We evaluate models under rollout budgets of 16, 32, 64, 128, and 256. Following Hochlehnert et al. [26], we repeat all experiments five times on MATH500 and ten times on AIME2024 and AIME2025, reporting the average performance across all runs to reduce the impact of randomness and improve the reliability of our conclusions. For reward assignment during rollouts, we use the final PRM score at each step as the reward for that step. The final answer is selected using weighted majority voting, where each trajectory is weighted by its final PRM score. We use these aggregation strategies since they have been shown to outperform other methods of aggregating trajectories to determine the final response [8]. See Appendix D.1 for experimental hyperparameters.

 doi:10.20944/preprints202506.0780.v1

4.2. Main Results

DORA is the most effective parallel search method. As shown in Figure 3 (a), DORA consistently achieves the highest accuracy across all policy models and rollout budgets on the MATH500, AIME2024, and AIME2025 benchmarks. This consistent superiority demonstrates DORA's advantage to make more efficient use of limited test-time compute compared to baseline strategies. To better understand this advantage, we further analyze the pass rate (the number of correct solutions among all sampled rollouts). As shown in Figure 3 (b), DORA consistently reaches more correct solutions than other baselines, highlighting its effectiveness in exploring a broader set of high-quality reasoning paths. Notably, the performance gap between DORA and REBASE widens as the rollout budget increases. We hypothesize that this is due to growing redundancy in sampled solutions: with more rollouts, a larger proportion of trajectories tend to converge to similar final solutions, making REBASE's solution-oriented allocation increasingly prone to overestimating certain reasoning directions. In contrast, DORA mitigates this issue by allocating rollouts at the direction level, allowing for more accurate resource allocation.

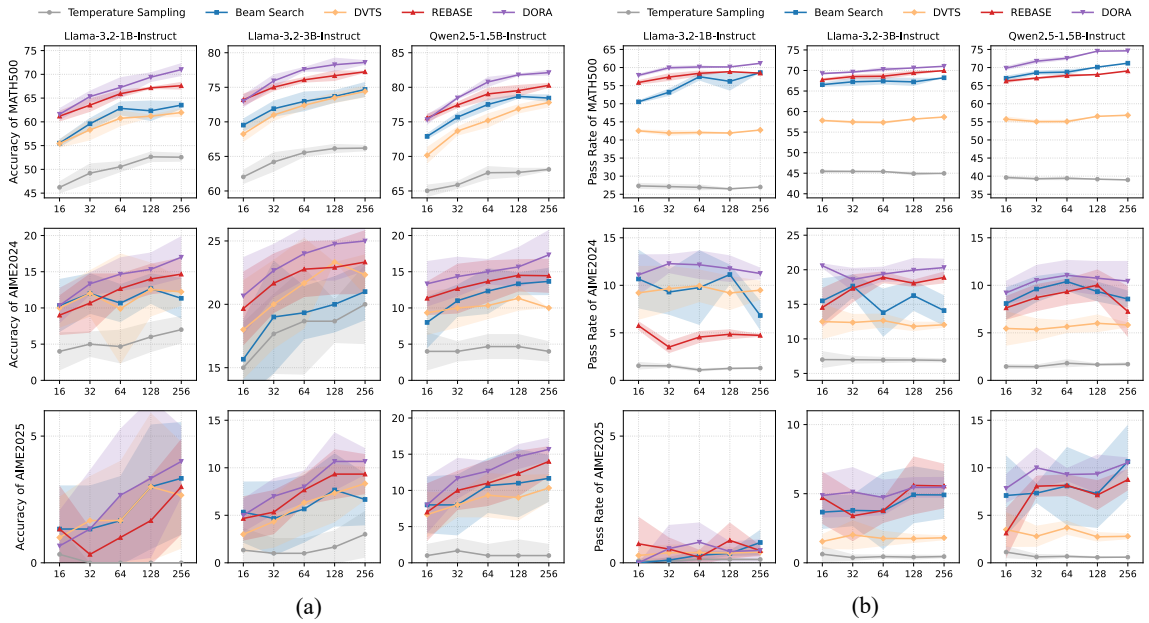

Figure 3. Accuracy and Pass rate comparison under various rollout budgets on MATH500, AIME2024, and AIME2025.

4.3. Analysis

DORA is compute-optimal. Considering that DORA introduces an additional semantic similarity step via an embedding model, we examine whether the associated computational overhead is justified by the performance gains. To this end, we follow Snell et al. [2], comparing the total FLOPs and inference latency of each method, accounting for the computational cost of the policy model, PRM, and embedding model. Table 1 reports both metrics alongside each method's accuracy. The results demonstrate that DORA is substantially more efficient than all baselines. Specifically, compared to the strongest baseline, REBASE at 256 rollouts, DORA achieves higher accuracy using only 64 rollouts, with a 3.5× reduction in total FLOPs and a 4× speedup in inference latency. These findings suggest that DORA achieves stronger performance with substantially less compute, demonstrating its effectiveness as the most efficient test-time search method.

 doi:10.20944/preprints202506.0780.v1

Table 1. Comparison of FLOPs and inference latency (s) of different methods on MATH500 using LLaMA-3.2-1B-Instruct. The best performance for each metric is highlighted in **bold**. Temperature Sampling is excluded due to its significantly lower accuracy.

Method	Rollout	FLOPs				Latency	Accuracy
		Policy Model	PRM	Embedding Model	Total		
Beam Search	256	3.58×10^{14}	2.50×10^{15}	0	2.86×10^{15}	345	63.6
DVTS	256	3.79×10^{14}	2.65×10^{15}	0	3.03×10^{15}	253	62.0
REBASE	256	3.88×10^{14}	2.72×10^{15}	0	3.11×10^{15}	490	67.4
DORA	64	8.45×10^{13}	5.92×10^{14}	2.16×10^{14}	$\mathbf{8.92 \times 10^{14}}$	**124**	**68.7**

DORA provides larger gains on harder problems. Figure 4 shows that while DORA remains the top-performing strategy across the entire MATH500 benchmark, the size of its advantage depends sharply on difficulty. On easier Level 1–2 problems, most methods perform well given moderate rollout budgets, so the accuracy curves for all methods converge closely. On the other hand, on harder Level 3–5 problems, the gap between DORA and solution-level methods widens steadily with budget, with DORA achieving a clear lead at higher rollout levels. We hypothesize that harder problems amplify DORA's strength as they typically require longer reasoning chains [27], which allows more opportunities for rollout allocation across search steps. As the number of allocation rounds increases, a principled strategy like DORA counld compound its advantage by continually prioritizing promising directions and avoiding wasted computation.

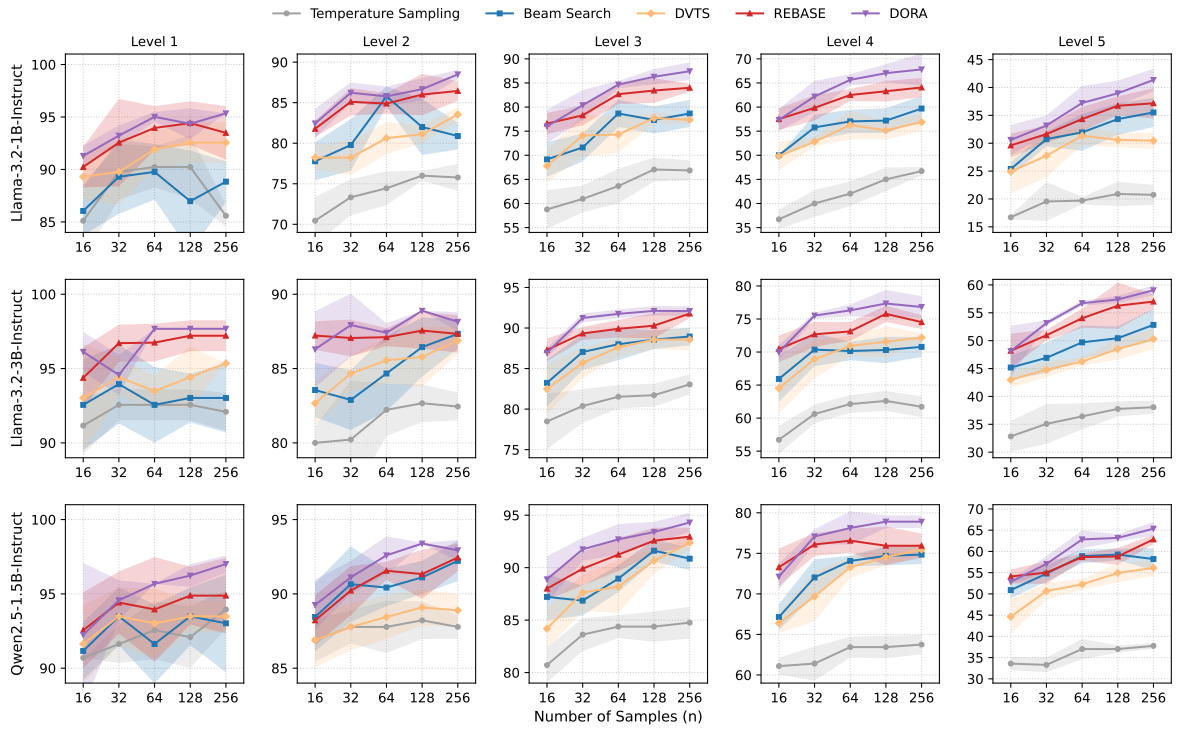

Figure 4. Comparison of method accuracy on MATH500 across different difficulty levels.

5. Related Work

LLM Test-Time Scaling.

Scaling LLM test-time compute is an effective way to improve performance [28]. Prior work has explored various strategies, including sampling-based methods with majority voting [29] and search-based techniques [30–32]. More recently, search algorithms such as breadth-first and depth-first search [33], and Monte Carlo Tree Search (MCTS) [34–37] have been applied to enhance reasoning. While these methods show promise, many rely on multi-step lookahead operations that are computationally expensive and limit practical scalability [2]. To improve efficiency, several studies have

 doi:10.20944/preprints202506.0780.v1

proposed parallel search strategies [2,3,8]. These strategies demonstrate the effectiveness of parallel search at inference time. However, how to allocate a fixed rollout budget most effectively during search remains underexplored.

Process Reward Models.

Process reward models (PRMs) have emerged as a powerful tool for improving the reasoning and problem-solving capabilities of large language models. By assigning rewards to intermediate steps, PRMs enable finer-grained evaluation and more effective guidance for multi-step reasoning. They have been shown effective in selecting low-error reasoning traces and providing reward signals for reinforcement-style optimization [38–40]. With their rapid development, benchmarks such as ProcessBench [22] and PRMBench [23] have been introduced to provide comprehensive evaluation protocols. Zhang et al. [21] further offer practical guidelines for training and deploying PRMs, releasing some of the strongest open-source PRMs to date, particularly for mathematical reasoning.

Mathematical Reasoning with LLMs.

Recent advances have significantly improved LLMs' performance on mathematical tasks, driven by both training-time and test-time techniques. Training-time methods include large-scale pretraining [41–43], supervised fine-tuning [44,45], and self-improvement via self-generated solutions [46–48]. Test-time approaches leverage CoT prompting [49,50], external tools [51,52], and self-verification [53] to enhance reasoning without changing model weights.

6. Conclusions

In this work, we formulate test-time search as a resource allocation problem and derive its optimal solution under a Bayesian framework. Our theoretical analysis offers a unified perspective that explains existing search methods as approximations under varying reward confidence. Furthermore, we find that solution-level allocation favors directions with more candidates and results in suboptimal use of test-time compute. To address this, we propose DORA, a direction-oriented allocation strategy that provably achieves optimality. Extensive experiments on three mathematical reasoning benchmarks demonstrate that DORA consistently improves performance while reducing compute cost. It achieves 3.5× fewer FLOPs and 4× lower latency compared to the strongest baseline REBASE. These results highlight DORA's ability to enhance both the effectiveness and efficiency of test-time inference.

Limitations. While our study focuses on scenarios where a process reward model (PRM) is available to evaluate partial trajectories, the underlying framework is not inherently tied to this specific signal. In principle, DORA can incorporate alternative forms of intermediate feedback, such as model confidence or likelihood-based heuristics, extending its applicability beyond PRM-supervised domains. Another limitation is that our theoretical analysis assumes a low-confidence setting, which may not fully capture the dynamics of confidence accumulation during multi-step reasoning. Adapting the allocation strategy to account for increasing confidence over time presents a promising direction for future work.

Appendix A. Details of Parallel Search Process

We present the detailed procedure of the Parallel Search Process in Algorithm A1.

 doi:10.20944/preprints202506.0780.v1

Algorithm A1 Parallel Search Process

Require: Input problem $x \sim X$, parameters (π, Q, O, V, N), step limit $T_{\max}$
Ensure: Final Answer

$A_0 \leftarrow \{\tau_j = x\}_{j=1}^{N}$ ▷ Initial active partial solutions
$T_{\text{final}} \leftarrow \emptyset$ ▷ Collected complete solutions
for $i = 0$ to $T_{\max} - 1$ **do**
  for all $\tau_j \in A_i$ **do**
    Sample action $a \sim \pi(\cdot \mid \tau_j)$
    $\tau_j \leftarrow \tau_j \circ a$
  end for
  $T_{\text{final}} \leftarrow T_{\text{final}} \cup \{\tau_j \in A_i \mid \texttt{<EOS>} \in \tau_j\}$ ▷ Add completed solutions
  $A_i \leftarrow A_i \setminus \{\tau_j \in A_i \mid \texttt{<EOS>} \in \tau_j\}$ ▷ Remove completed solutions
  if $|T_{\text{final}}| \geq N$ **then**
    break
  end if
  Compute PRM scores: $R_j \leftarrow Q(\tau_j)$ for each $\tau_j \in A_i$
  Compute rollout allocation: $B_j \leftarrow O(\boldsymbol{R})_j$, where $\boldsymbol{R} = \{R_1, \ldots, R_{|A_i|}\}$
  $A_{i+1} \leftarrow \emptyset$
  for $j = 1$ to $|A_i|$ **do**
    Add B_j copies of τ_j to A_{i+1}
  end for
end for
return $V(T_{\text{final}})$ ▷ Select final answer from complete solutions

Appendix B. Proof Section

Appendix B.1. Proof of Proposition 1

Let $p_i \sim \text{Beta}(\kappa w_i, \kappa(1 - w_i))$, where $w_i \in (0, 1)$ is the normalized PRM score for candidate τ_i. Allocating B_i rollouts to candidate i, the expected failure probability is

$$\mathbb{E}\left[\prod_{i=1}^{k}(1 - p_i)^{B_i}\right] = \prod_{i=1}^{k}\mathbb{E}\left[(1 - p_i)^{B_i}\right].$$

Using the identity for Beta-distributed p_i, we have:

$$\mathbb{E}\left[(1 - p_i)^{B_i}\right] = \prod_{r=0}^{B_i-1}\frac{\kappa(1 - w_i) + r}{\kappa + r}.$$

Taking the negative logarithm of the success probability, the equivalent optimization problem becomes:

$$\min_{\sum B_i = N}\sum_{i=1}^{k}\sum_{r=0}^{B_i-1} - \log\left(1 - \frac{\kappa w_i}{\kappa + r}\right).$$

Using the identity $\sum_{r=0}^{n-1}\frac{1}{\kappa+r} = \psi(\kappa + n) - \psi(\kappa)$, where ψ is the digamma function, the objective simplifies to:

$$L(\boldsymbol{B}) = \sum_{i=1}^{k}\kappa w_i[\psi(\kappa + B_i) - \psi(\kappa)].$$

Relaxing $B_i \in \mathbb{N}$ to $B_i \in \mathbb{R}_{\geq 0}$, we apply the method of Lagrange multipliers with constraint $\sum_i B_i = N$. The partial derivatives yield:

$$\frac{\partial L}{\partial B_i} = \kappa w_i \cdot \psi'(\kappa + B_i),$$

where ψ' is the trigamma function. The KKT condition implies that at optimality:

$$\kappa w_i \cdot \psi'(\kappa + B_i) = \lambda, \quad \text{for all } i \text{ with } B_i > 0, \quad \text{and} \quad \sum B_i = N.$$

 doi:10.20944/preprints202506.0780.v1

We now analyze three asymptotic regimes of κ:

Case 1: Fixed finite $\kappa > 0$ Using the approximation $\psi'(\kappa + B_i) \approx \frac{1}{\kappa + B_i}$ when $\kappa + B_i \gg 1$, the optimality condition becomes:

$$\frac{\kappa w_i}{\kappa + B_i} \approx \lambda \quad \Rightarrow \quad B_i^\star \approx \frac{\kappa w_i}{\lambda} - \kappa.$$

Summing both sides over i and enforcing $\sum_i B_i = N$, we solve for $\lambda \approx \kappa/(N + k\kappa)$, yielding:

$$\boxed{B_i^\star \approx (N + k\kappa) w_i - \kappa.}$$

Case 2: $\kappa \to \infty$ In this regime, the Beta prior becomes increasingly concentrated at $p_i = w_i$. Hence,

$$\mathbb{E}[(1 - p_i)^{B_i}] \to (1 - w_i)^{B_i}, \quad \text{and} \quad \mathbb{E}\left[\prod_{i=1}^{k} (1 - p_i)^{B_i}\right] \to \prod_{i=1}^{k} (1 - w_i)^{B_i}.$$

To minimize failure probability, we solve:

$$\min_{\sum B_i = N} \sum_{i=1}^{k} B_i \log(1 - w_i).$$

Since $\log(1 - w_i) < 0$, this is minimized by allocating all rollouts to the candidate with the largest w_i, i.e.,

$$O^\star(\boldsymbol{w})_i = \begin{cases} N, & \text{if } i = \arg\max_j w_j, \\ 0, & \text{otherwise.} \end{cases}$$

Case 3: $\kappa \to 0$ In this regime, the Beta distribution becomes highly uncertain:

$$p_i \sim \text{Beta}(\kappa w_i, \kappa(1 - w_i)) \xrightarrow[\kappa \to 0]{} \begin{cases} 1, & \text{with probability } w_i, \\ 0, & \text{with probability } 1 - w_i. \end{cases}$$

Hence,

$$\mathbb{E}\left[(1 - p_i)^{B_i}\right] \to \begin{cases} 1 - w_i, & \text{if } B_i > 0, \\ 1, & \text{if } B_i = 0. \end{cases}$$

Thus, the expected failure probability becomes:

$$\prod_{i=1}^{k} \mathbb{E}\left[(1 - p_i)^{B_i}\right] \to \prod_{i : B_i > 0} (1 - w_i),$$

which depends only on whether a candidate receives at least one rollout, not how many. To minimize failure, we must select a subset $S \subseteq \{1, \ldots, k\}$ with $|S| \leq N$ such that:

$$\prod_{i \in S} (1 - w_i)$$

is minimized. This is achieved by choosing the top-$s = \min(k, N)$ candidates with the largest w_i. Then the optimal allocation is:

$$O^\star(\boldsymbol{w})_i = \begin{cases} 1, & \text{if } i \in \text{Top-}s \text{ of } w_i, \\ 0, & \text{otherwise,} \end{cases} \quad \text{with remaining } N - s \text{ rollouts arbitrarily assigned.}$$

 doi:10.20944/preprints202506.0780.v1

Appendix B.2. Proof of Proposition 2

In the $\kappa \to 0$ regime, Proposition 1 shows that the expected success probability is maximized by the solution:

$$B_i \propto w_i, \quad \text{where } w_i = \frac{e^{R_i}}{\sum_j e^{R_j}}.$$

This corresponds to maximizing the log-utility objective:

$$\mathcal{L} = \sum_{i=1}^{k} w_i \log B_i.$$

To analyze the effect of structural redundancy, we group candidate solutions into g reasoning directions. Let direction j contain k_j candidates, each with identical score R_j, and index set $\mathcal{E}_j$.

The optimal direction-aware allocation follows:

$$Q_j := \frac{e^{R_j}}{\sum_{l=1}^{g} e^{R_l}}, \qquad B_j^{(\text{direction})} := N \cdot Q_j.$$

The corresponding log-utility is:

$$\mathcal{L}^{(\text{dir})} = \sum_{j=1}^{g} Q_j \log B_j^{(\text{direction})} = \log N + \sum_{j=1}^{g} Q_j \log Q_j.$$

REBASE assigns each candidate $i \in \mathcal{E}_j$ rollout weight:

$$w_i = \frac{e^{R_j}}{\sum_{l=1}^{g} k_l e^{R_l}}, \quad \text{so} \quad B_j^{(\text{solution})} = \sum_{i \in \mathcal{E}_j} N w_i = N \cdot \frac{k_j e^{R_j}}{\sum_{l=1}^{g} k_l e^{R_l}}.$$

This induces a direction-level distribution:

$$\hat{Q}_j := \frac{k_j e^{R_j}}{\sum_{l=1}^{g} k_l e^{R_l}}.$$

The resulting utility is:

$$\mathcal{L}^{(\text{sol})} = \sum_{j=1}^{g} Q_j \log B_j^{(\text{solution})} = \log N + \sum_{j=1}^{g} Q_j \log \hat{Q}_j.$$

The gap in log-utility is:

$$\mathcal{L}^{(\text{dir})} - \mathcal{L}^{(\text{sol})} = \sum_{j=1}^{g} Q_j \log \frac{Q_j}{\hat{Q}_j} = \mathrm{KL}(Q \,\|\, \hat{Q}) \geq 0.$$

Equality holds if and only if $Q_j = \hat{Q}_j$ for all j, i.e.,

$$\frac{e^{R_j}}{\sum_l e^{R_l}} = \frac{k_j e^{R_j}}{\sum_l k_l e^{R_l}} \quad \Rightarrow \quad k_j = k \text{ for all } j.$$

Thus, the solution-level allocation is suboptimal unless all reasoning directions contain the same number of candidate solutions.

Appendix B.3. Proof of Theorem 1

Assume candidate solutions are partitioned into g reasoning directions, where direction $j \in \{1, \ldots, g\}$ contains k_j candidates indexed by $\mathcal{E}_j$, and all candidates in $\mathcal{E}_j$ share the same PRM score R_j.

 doi:10.20944/preprints202506.0780.v1

Under REBASE, softmax is computed at the solution level:

$$\tilde{q}_i = \frac{e^{R_j}}{\sum_{l=1}^{g} k_l e^{R_l}}, \quad \text{for } i \in \mathcal{E}_j.$$

Aggregating across each direction yields the induced direction-level distribution:

$$\hat{Q}_j^{\text{REBASE}} = \sum_{i \in \mathcal{E}_j} \tilde{q}_i = \frac{k_j e^{R_j}}{\sum_{l=1}^{g} k_l e^{R_l}}.$$

To eliminate the bias from uneven candidate counts k_j, DORA reweights each $\tilde{q}_i$ by the inverse of its cluster size:

$$\hat{q}_i = \frac{\tilde{q}_i}{k_j}, \quad \text{for } i \in \mathcal{E}_j.$$

The normalization constant becomes:

$$Z = \sum_{i=1}^{k} \hat{q}_i = \sum_{j=1}^{g} \sum_{i \in \mathcal{E}_j} \frac{\tilde{q}_i}{k_j} = \sum_{j=1}^{g} \frac{k_j e^{R_j}}{k_j \sum_{l=1}^{g} k_l e^{R_l}} = \frac{\sum_{j=1}^{g} e^{R_j}}{\sum_{l=1}^{g} k_l e^{R_l}}.$$

Normalizing gives the final corrected weight:

$$\hat{q}_i^{\text{final}} = \frac{\hat{q}_i}{Z} = \frac{e^{R_j}}{k_j \sum_{l=1}^{g} e^{R_l}}, \quad \text{for } i \in \mathcal{E}_j.$$

Aggregating over direction j, the direction-level allocation becomes:

$$\hat{Q}_j^{\text{final}} = \sum_{i \in \mathcal{E}_j} \hat{q}_i^{\text{final}} = k_j \cdot \frac{e^{R_j}}{k_j \sum_{l=1}^{g} e^{R_l}} = \frac{e^{R_j}}{\sum_{l=1}^{g} e^{R_l}} = Q_j.$$

Thus, the final allocation satisfies

$$\sum_{i \in \mathcal{E}_j} B_i \propto Q_j,$$

which exactly matches the optimal direction-level allocation given in Eq. 9.

Appendix C. Details of Beta Distribution

The Beta distribution is a standard choice for modeling random variables on the unit interval, and its parameters $(\alpha, \beta) = (\kappa w_i, \kappa(1 - w_i))$ are interpretable: the mean is $\mathbb{E}[p_i] = w_i$, and the variance is inversely related to κ. Specifically:

- When κ is small, the distribution is diffuse and uncertain.
- When κ is large, the distribution is sharply peaked around w_i, indicating high confidence.

Figure A1 visualizes the effect of different κ values with w_i fixed at 0.7.

 doi:10.20944/preprints202506.0780.v1

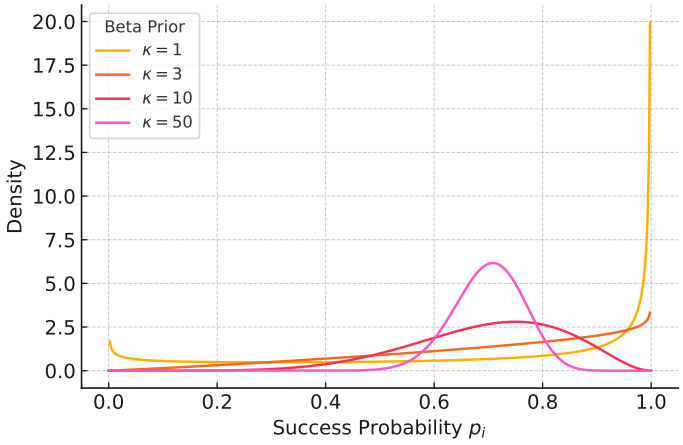

Figure A1. Effect of the concentration parameter κ on the Beta prior. All curves are plotted with fixed mean $w_i = 0.7$. Larger κ yields a more concentrated prior around w_i, while smaller κ reflects greater uncertainty.

Appendix D. Implementation Details

Appendix D.1. Experimental Hyperparameters

All experiments use temperature sampling with `temperature` = 0.8 and `top_p` = 1.0. We set the token limit to 256 per step and 2048 tokens in total for each solution. For Beam Search and DVTS, we use a beam width of 4 following Snell et al. [2]. For REBASE, we set its T_b to 0.1, consistent with its original implementation. For DORA, we employ the open-source BGE-M3 embedding model [54] to compute semantic similarity between trajectories, chosen for its lightweight architecture, strong empirical performance, and ability to handle long input sequences. We set the T_b for quality scores to 0.1 (matching REBASE), and the semantic similarity temperature T_s to 0.01. All experiments are executed in parallel on a cluster with 32 NVIDIA A100 GPUs (40G), where each individual run is allocated to a single GPU.

Appendix D.2. Details of Prompt

Following Beeching et al. [8], we employ the prompt below for LLM mathematical reasoning:

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

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

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

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

...
## Step 2: ...

Regardless of problem complexity, always conclude with:
Therefore, the final answer is: \boxed{answer}.
```

References

1. Brown, B.; Juravsky, J.; Ehrlich, R.; Clark, R.; Le, Q.V.; Ré, C.; Mirhoseini, A. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787* **2024**.

 doi:10.20944/preprints202506.0780.v1

2. Snell, C.; Lee, J.; Xu, K.; Kumar, A. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314* **2024**.
3. Wu, Y.; Sun, Z.; Li, S.; Welleck, S.; Yang, Y. Inference scaling laws: An empirical analysis of compute-optimal inference for LLM problem-solving. In Proceedings of the The Thirteenth International Conference on Learning Representations, 2025.
4. Qwen Team. QwQ: Reflect Deeply on the Boundaries of the Unknown, 2024.
5. Kimi Team.; Du, A.; Gao, B.; Xing, B.; Jiang, C.; Chen, C.; Li, C.; Xiao, C.; Du, C.; Liao, C.; et al. Kimi k1.5: Scaling Reinforcement Learning with LLMs. *arXiv preprint arXiv:2501.12599* **2025**.
6. DeepSeek-AI.; Guo, D.; Yang, D.; Zhang, H.; Song, J.; Zhang, R.; Xu, R.; Zhu, Q.; Ma, S.; Wang, P.; et al. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. *arXiv preprint arXiv:2501.12948* **2025**.
7. Chen, L.; Davis, J.Q.; Hanin, B.; Bailis, P.; Stoica, I.; Zaharia, M.A.; Zou, J.Y. Are more llm calls all you need? towards the scaling properties of compound ai systems. *Advances in Neural Information Processing Systems* **2024**, *37*, 45767–45790.
8. Beeching, E.; Tunstall, L.; Rush, S. Scaling test-time compute with open models, 2024.
9. Liu, R.; Gao, J.; Zhao, J.; Zhang, K.; Li, X.; Qi, B.; Ouyang, W.; Zhou, B. Can 1B LLM Surpass 405B LLM? Rethinking Compute-Optimal Test-Time Scaling. *arXiv preprint arXiv:2502.06703* **2025**.
10. Setlur, A.; Rajaraman, N.; Levine, S.; Kumar, A. Scaling test-time compute without verification or rl is suboptimal. *arXiv preprint arXiv:2502.12118* **2025**.
11. Bi, Z.; Han, K.; Liu, C.; Tang, Y.; Wang, Y. Forest-of-thought: Scaling test-time compute for enhancing LLM reasoning. *arXiv preprint arXiv:2412.09078* **2024**.
12. Hooper, C.; Kim, S.; Moon, S.; Dilmen, K.; Maheswaran, M.; Lee, N.; Mahoney, M.W.; Shao, S.; Keutzer, K.; Gholami, A. Ets: Efficient tree search for inference-time scaling. *arXiv preprint arXiv:2502.13575* **2025**.
13. Hendrycks, D.; Burns, C.; Kadavath, S.; Arora, A.; Basart, S.; Tang, E.; Song, D.; Steinhardt, J. Measuring Mathematical Problem Solving With the MATH Dataset. In Proceedings of the Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual, 2021.
14. AI-MO. AIME 2024, 2024.
15. Jiang, J.; Chen, Z.; Min, Y.; Chen, J.; Cheng, X.; Wang, J.; Tang, Y.; Sun, H.; Deng, J.; Zhao, W.X.; et al. Technical Report: Enhancing LLM Reasoning with Reward-guided Tree Search. *arXiv preprint arXiv:2411.11694* **2024**.
16. Wang, P.; Li, L.; Shao, Z.; Xu, R.; Dai, D.; Li, Y.; Chen, D.; Wu, Y.; Sui, Z. Math-Shepherd: Verify and Reinforce LLMs Step-by-step without Human Annotations. In Proceedings of the Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), 2024, pp. 9426–9439.
17. Luo, L.; Liu, Y.; Liu, R.; Phatale, S.; Guo, M.; Lara, H.; Li, Y.; Shu, L.; Zhu, Y.; Meng, L.; et al. Improve mathematical reasoning in language models by automated process supervision. *arXiv preprint arXiv:2406.06592* **2024**.
18. Wang, Z.; Li, Y.; Wu, Y.; Luo, L.; Hou, L.; Yu, H.; Shang, J. Multi-step Problem Solving Through a Verifier: An Empirical Analysis on Model-induced Process Supervision. In Proceedings of the Findings of the Association for Computational Linguistics: EMNLP 2024, 2024, pp. 7309–7319.
19. Setlur, A.; Nagpal, C.; Fisch, A.; Geng, X.; Eisenstein, J.; Agarwal, R.; Agarwal, A.; Berant, J.; Kumar, A. Rewarding Progress: Scaling Automated Process Verifiers for LLM Reasoning. In Proceedings of the The Thirteenth International Conference on Learning Representations, 2025.
20. Lee, J.H.; Yang, J.Y.; Heo, B.; Han, D.; Kim, K.; Yang, E.; Yoo, K.M. Token-Supervised Value Models for Enhancing Mathematical Problem-Solving Capabilities of Large Language Models. In Proceedings of the The Thirteenth International Conference on Learning Representations, 2025.
21. Zhang, Z.; Zheng, C.; Wu, Y.; Zhang, B.; Lin, R.; Yu, B.; Liu, D.; Zhou, J.; Lin, J. The Lessons of Developing Process Reward Models in Mathematical Reasoning. *arXiv preprint arXiv:2501.07301* **2025**.
22. Zheng, C.; Zhang, Z.; Zhang, B.; Lin, R.; Lu, K.; Yu, B.; Liu, D.; Zhou, J.; Lin, J. Processbench: Identifying process errors in mathematical reasoning. *arXiv preprint arXiv:2412.06559* **2024**.
23. Song, M.; Su, Z.; Qu, X.; Zhou, J.; Cheng, Y. PRMBench: A Fine-grained and Challenging Benchmark for Process-Level Reward Models. *arXiv preprint arXiv:2501.03124* **2025**.
24. AI, M. Llama 3.2: Multilingual Instruction-Tuned Language Models. https://huggingface.co/meta-llama/Llama-3.2-1B-Instruct, 2024. Accessed: 2025-05-14.
25. Yang, A.; Yang, B.; Zhang, B.; Hui, B.; Zheng, B.; Yu, B.; Li, C.; Liu, D.; Huang, F.; Wei, H.; et al. Qwen2.5 Technical Report. *arXiv preprint arXiv:2412.15115* **2024**.

 doi:10.20944/preprints202506.0780.v1

26. Hochlehnert, A.; Bhatnagar, H.; Udandarao, V.; Albanie, S.; Prabhu, A.; Bethge, M. A sober look at progress in language model reasoning: Pitfalls and paths to reproducibility. *arXiv preprint arXiv:2504.07086* **2025**.
27. Wu, Y.; Wang, Y.; Du, T.; Jegelka, S.; Wang, Y. When More is Less: Understanding Chain-of-Thought Length in LLMs. *arXiv preprint arXiv:2502.07266* **2025**.
28. OpenAI. Learning to Reason with LLMs, 2024.
29. Wang, X.; Wei, J.; Schuurmans, D.; Le, Q.V.; Chi, E.H.; Narang, S.; Chowdhery, A.; Zhou, D. Self-Consistency Improves Chain of Thought Reasoning in Language Models. In Proceedings of the The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023. OpenReview.net, 2023.
30. Xie, Y.; Kawaguchi, K.; Zhao, Y.; Zhao, J.X.; Kan, M.Y.; He, J.; Xie, M. Self-Evaluation Guided Beam Search for Reasoning. In Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), 2023, Vol. 36, pp. 41618–41650.
31. Khanov, M.; Burapacheep, J.; Li, Y. ARGS: Alignment as Reward-Guided Search. In Proceedings of the International Conference on Learning Representations (ICLR), 2024.
32. Wan, Z.; Feng, X.; Wen, M.; Mcaleer, S.M.; Wen, Y.; Zhang, W.; Wang, J. AlphaZero-Like Tree-Search can Guide Large Language Model Decoding and Training. In Proceedings of the International Conference on Machine Learning (ICML), 2024, Vol. 235, pp. 49890–49920.
33. Yao, S.; Yu, D.; Zhao, J.; Shafran, I.; Griffiths, T.; Cao, Y.; Narasimhan, K. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. In Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), 2023, Vol. 36, pp. 11809–11822.
34. Ma, Q.; Zhou, H.; Liu, T.; Yuan, J.; Liu, P.; You, Y.; Yang, H. Let's reward step by step: Step-Level reward model as the Navigators for Reasoning. *arXiv preprint arXiv:2310.10080* **2023**.
35. Li, Y.; Lin, Z.; Zhang, S.; Fu, Q.; Chen, B.; Lou, J.G.; Chen, W. Making large language models better reasoners with step-aware verifier. *arXiv preprint arXiv:2206.02336* **2022**.
36. Liu, J.; Cohen, A.; Pasunuru, R.; Choi, Y.; Hajishirzi, H.; Celikyilmaz, A. Don't throw away your value model! Generating more preferable text with Value-Guided Monte-Carlo Tree Search decoding. *arXiv preprint arXiv:2309.15028* **2023**.
37. Choi, S.; Fang, T.; Wang, Z.; Song, Y. KCTS: Knowledge-Constrained Tree Search Decoding with Token-Level Hallucination Detection. In Proceedings of the Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing; Bouamor, H.; Pino, J.; Bali, K., Eds., Singapore, 2023; pp. 14035–14053. https://doi.org/10.18653/v1/2023.emnlp-main.867.
38. Uesato, J.; Kushman, N.; Kumar, R.; Song, F.; Siegel, N.; Wang, L.; Creswell, A.; Irving, G.; Higgins, I. Solving math word problems with process-and outcome-based feedback. *arXiv preprint arXiv:2211.14275* **2022**.
39. Polu, S.; Sutskever, I. Generative language modeling for automated theorem proving. *arXiv preprint arXiv:2009.03393* **2020**.
40. Gudibande, A.; Wallace, E.; Snell, C.; Geng, X.; Liu, H.; Abbeel, P.; Levine, S.; Song, D. The false promise of imitating proprietary llms. *arXiv preprint arXiv:2305.15717* **2023**.
41. OpenAI. GPT-4 Technical Report. *arXiv preprint arXiv:2303.08774* **2023**.
42. Azerbayev, Z.; Schoelkopf, H.; Paster, K.; Santos, M.D.; McAleer, S.M.; Jiang, A.Q.; Deng, J.; Biderman, S.; Welleck, S. Llemma: An Open Language Model for Mathematics. In Proceedings of the International Conference on Learning Representations (ICLR), 2024.
43. Shao, Z.; Wang, P.; Zhu, Q.; Xu, R.; Song, J.; Bi, X.; Zhang, H.; Zhang, M.; Li, Y.; Wu, Y.; et al. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models. *arXiv preprint arXiv:2402.03300* **2024**.
44. Luo, H.; Sun, Q.; Xu, C.; Zhao, P.; Lou, J.; Tao, C.; Geng, X.; Lin, Q.; Chen, S.; Zhang, D. Wizardmath: Empowering mathematical reasoning for large language models via reinforced evol-instruct. *arXiv preprint arXiv:2308.09583* **2023**.
45. Tang, Z.; Zhang, X.; Wang, B.; Wei, F. MathScale: Scaling Instruction Tuning for Mathematical Reasoning. In Proceedings of the International Conference on Machine Learning (ICML), 2024, Vol. 235, pp. 47885–47900.
46. Zelikman, E.; Wu, Y.; Mu, J.; Goodman, N. STaR: Bootstrapping Reasoning With Reasoning. In Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), 2022, Vol. 35, pp. 15476–15488.
47. Gulcehre, C.; Paine, T.L.; Srinivasan, S.; Konyushkova, K.; Weerts, L.; Sharma, A.; Siddhant, A.; Ahern, A.; Wang, M.; Gu, C.; et al. Reinforced Self-Training (ReST) for Language Modeling. *arXiv preprint arXiv:2308.08998* **2023**.
48. Setlur, A.; Garg, S.; Geng, X.; Garg, N.; Smith, V.; Kumar, A. RL on Incorrect Synthetic Data Scales the Efficiency of LLM Math Reasoning by Eight-Fold. *arXiv preprint arXiv:2406.14532* **2024**.

 doi:10.20944/preprints202506.0780.v1

49. Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Ichter, B.; Xia, F.; Chi, E.H.; Le, Q.V.; Zhou, D. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In Proceedings of the NeurIPS, 2022.
50. Zhao, S.; Yuan, J.; Yang, G.; Naseem, U. Can Pruning Improve Reasoning? Revisiting Long-CoT Compression with Capability in Mind for Better Reasoning. *arXiv preprint arXiv:2505.14582* **2025**.
51. Gao, L.; Madaan, A.; Zhou, S.; Alon, U.; Liu, P.; Yang, Y.; Callan, J.; Neubig, G. PAL: Program-aided Language Models. In Proceedings of the International Conference on Machine Learning (ICML), 2023, Vol. 202, pp. 10764–10799.
52. Chen, W.; Ma, X.; Wang, X.; Cohen, W.W. Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks. *Transactions on Machine Learning Research (TMLR)* **2023**.
53. Weng, Y.; Zhu, M.; Xia, F.; Li, B.; He, S.; Liu, S.; Sun, B.; Liu, K.; Zhao, J. Large language models are better reasoners with self-verification. In Proceedings of the Findings of the Association for Computational Linguistics: EMNLP 2023, 2023, pp. 2550–2575.
54. Chen, J.; Xiao, S.; Zhang, P.; Luo, K.; Lian, D.; Liu, Z. BGE M3-Embedding: Multi-Lingual, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation, 2024, [arXiv:cs.CL/2402.03216].