A Probabilistic Inference Approach to Inference-Time Scaling of LLMs using Particle-Based Monte Carlo Methods

Isha Puri [1] **Shivchander Sudalairaj** [2] **Guangxuan Xu** [2] **Kai Xu** [2] **Akash Srivastava** [2]

Abstract

Large language models (LLMs) have achieved significant performance gains via scaling up model sizes and/or data. However, recent evidence suggests diminishing returns from such approaches, motivating scaling the computation spent at inference time. Existing inference-time scaling methods, usually with reward models, cast the task as a search problem, which tends to be vulnerable to reward hacking as a consequence of approximation errors in reward models. In this paper, we instead cast inference-time scaling as a probabilistic inference task and leverage sampling-based techniques to explore the typical set of the state distribution of a state-space model with an approximate likelihood, rather than optimize for its mode directly. We propose a novel inference-time scaling approach by adapting particle-based Monte Carlo methods to this task. Our empirical evaluation demonstrates that our methods have a 4–16x better scaling rate over our deterministic search counterparts on various challenging mathematical reasoning tasks. Using our approach, we show that Qwen2.5-Math-1.5B-Instruct can surpass GPT-4o accuracy in only 4 rollouts, while Qwen2.5-Math-7B-Instruct scales to o1 level accuracy in only 32 rollouts. Our work not only presents an effective method to inference-time scaling, but also connects the rich literature in probabilistic inference with inference-time scaling of LLMs to develop more robust algorithms in future work. Code and further information is available at https://probabilistic-inference-scaling.github.io/ .

1. Introduction

Large language models (LLMs) have demonstrated remarkable improvements in performance through scaling up model sizes and/or data. While frontier models have relied heavily on larger datasets and an ever-increasing number of learnable parameters (Kaplan et al., 2020; Snell et al., 2024), smaller LLMs have successfully leveraged domain-specific data to match the performance of larger, general-purpose models (Sudalairaj et al., 2024; Pareja et al., 2024). However, recent reports indicate plateaus in performance gains through such scaling methods. Consequently, inference-time (aka compute-time / test-time) scaling has emerged as a promising alternative to improve model performance (Beeching et al., 2024). Proprietary models like OpenAI's o1 (OpenAI et al., 2024) and o3 have demonstrated the benefits of allocating more computation resources at inference time, particularly for complex reasoning and math tasks. These inference-time scaling techniques not only enhance model capability but also allow smaller models to achieve performance levels comparable to their larger counterparts, making advanced AI more accessible for low-resource devices.

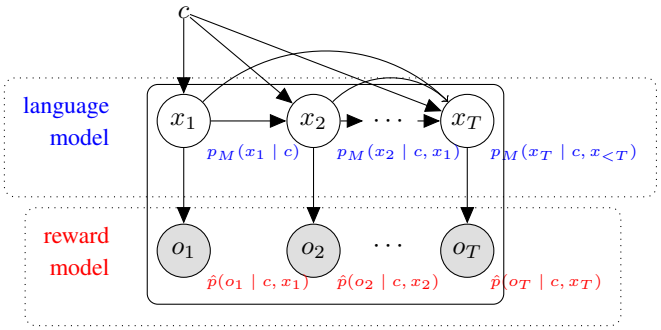
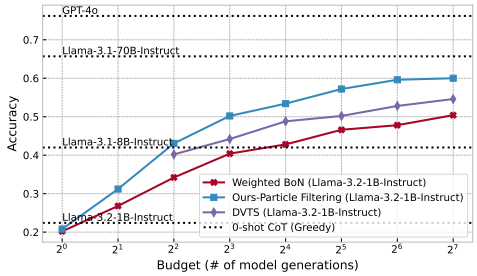


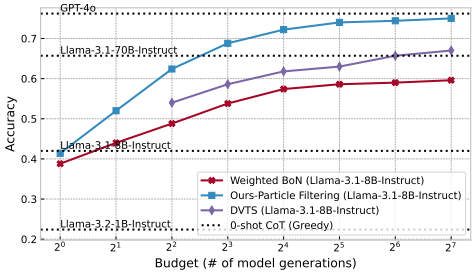
Figure 1. State-space model for inference-time scaling. c is a prompt, $x_1, \ldots, x_T$ are sequence of partial LLM outputs and $o_1, \ldots, o_T$ are the "observed" acceptance. We cast inference-time scaling as to estimate the latent states conditioned on $o_t = 1$ for $t = 1, 2, \ldots, T$, i.e. all being accepted.

Recent work (Lightman et al., 2023a) has framed inference-time scaling as a search problem guided by a process reward model (PRM). This perspective has led to the successful application of classic algorithms such as best-of-n (BoN; Brown et al., 2024), beam search (Zhou et al., 2024; Snell et al., 2024), and Monte Carlo tree search (MCTS; Guan et al., 2025), which refine model outputs by systematically exploring a broader search space. This process is sometimes referred to as "thinking/reasoning".

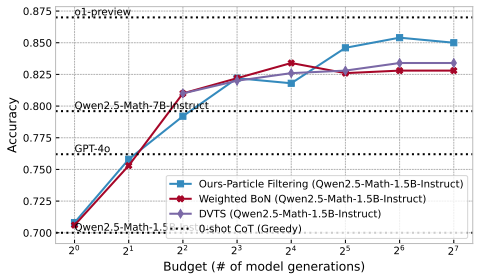
However, we argue that a search-based formulation becomes problematic when the reward model is imperfect—an inherent issue since these models are only approximations of an unknown true classification or preference function. Empirically, this often leads to reward hacking, where the final output is optimized to score well according to the reward model



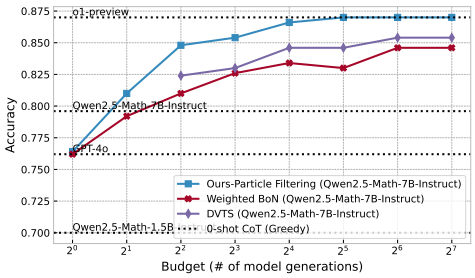
(a) Llama-3.2-1B-Instruct



(b) Llama-3.1-8B-Instruct



(c) Qwen2.5-Math-1.5B-Instruct



(d) Qwen2.5-Math-7B-Instruct

Figure 2. Performance of PF compared to other inference-time scaling methods across different model families. Figure 2a and Figure 2b demonstrate results for the Llama-3 family, where PF outperforms WBoN and DVTS in both cases and approaches the performance of much larger models like Llama-3.1-70B and even GPT-4o . Figure 2c and Figure 2d show results for the Qwen family, where PF achieves superior scaling against baslines, enabling the smaller model Qwen2.5-Math-1.5B-Instruct to surpass GPT-4o in performance within a limited compute budget. Larger Qwen2.5-Math-7B-Instruct model efficiently scale to match o1-preview performance on MATH500.

but fails to be useful and/or correct (Snell et al., 2024).

In this paper, we propose a shift in perspective by framing inference-time scaling as a probabilistic inference task. Unlike search-based methods that seek the mode of the reward model's distribution, we leverage sampling-based techniques to explore the typical set, which is more likely to overlap with the ground truth. This approach reduces reliance on potentially flawed reward models, as probabilistic inference naturally balances exploitation and exploration by trusting the reward model only up-to a certain probability (Andrieu et al., 2010). More specifically, unlike existing search-based methods in inference-time scaling, our probabilistic approach to scaling strikes a unique balance between exploration and exploitation. If the search process discovers a partial solution with a high process reward score, the next step will resample that solution more heavily but will typically not have it *completely* dominate the next step of particles, allowing for more diverse options to still continue their exploration.

The idea of using more computation to refine results is a fundamental feature of many classic probabilistic inference methods. For instance, Markov chain Monte Carlo (MCMC) methods improve inference asymptotically with more iterations, while particle-based Monte Carlo methods enhance accuracy as the number of particles increases.

Building on this principle, we introduce a novel approach to inference-time scaling by adapting particle-based Monte Carlo algorithms from probabilistic inference. Our method explicitly accounts for imperfections in reward models by maintaining a diverse set of candidates within the solution space. By iteratively updating their weights based on observed evidence (approximate reward), our approach ensures robust scaling even when the reward model is imperfect.

Our key contributions are as follows.

1. We formulate inference-time scaling as probabilistic inference over a state space model (SSM) jointly defined by a language model (transition kernel) and a process reward model (emission model), which enables direct application of probabilistic inference methods.

2. We propose inference-time scaling algorithms based on the particle filtering (PF) algorithm, which is robust to imperfection in reward modeling. We study its scaling performance and the effective temperature in LLM generation and how to optimally allocate computation budget over its multi-iteration and parallel extensions.

3. We study ways to use PRMs and propose a more robust and performant way to obtain rewards for partial answers which we refer to as model-based aggregation.

4. We demonstrate that the proposed methods have 4–16x faster scaling speed than previous methods based on a search formulation on the MATH500 and AIME

2024 datasets, with small language models in the Llama and Qwen families. We show that PF can scale Qwen2.5-Math-1.5B-Instruct to surpasses GPT-4o accuracy with only a budget of 4 and scale Qwen2.5-Math-7B-Instruct to o1 accuracy with a budget of 32.

2. Related Work

Process reward models (PRMs) aim to provide more granular feedback by evaluating intermediate steps rather than only final outputs. They are trained via process supervision, a training approach where models receive feedback on each intermediate step of their reasoning process rather than only on the final outcome. Lightman et al. (2023a) propose a step-by-step verification approach to PRMs, improving the reliability of reinforcement learning. DeepSeek PRM (Wang et al., 2024) uses Mistral to annotate training data for PRMs . Zhang et al. (2025b) introduces Qwen-PRM, which combines both Monte Carlo estimation and model/human annotation approach to prepare training data for a PRM. PRIME (Cui et al., 2025) proposes to train an outcome reward model (ORM) using an implicit reward objective. The paper shows that implicit reward objective directly learns a Q-function that provides rewards for each token, which can be leveraged to create process-level reward signal. This process eliminates the need for any process labels, and reaches competitive performance on PRM benchmarks.

Inference-time scaling has been a key training-free strategy for enhancing LLM performance. Brown et al. (2024) explores a best-of-N (BoN) decoding strategy, demonstrating improvements in output quality through selective refinement. (Snell et al., 2024) provides insights into how scaling compute resources can yield better inference efficiency from a compute optimality perspective. While not implementing full Monte Carlo tree search (MCTS), Zhou et al. (2024) explores a tree-search-like approach within language models. Additionally, Guan et al. (2025) introduces rSTAR, a method that combines MCTS for data generation and training to improve mathematical reasoning. Beeching et al. (2024) discusses beam search and dynamic variable-time search (DVTS) as inference-time scaling techniques to improve open-source LLMs. DVTS works by running multiple independent subtrees in parallel so to avoid all leaves stuck in local minima.

Particle-based Monte Carlo methods are powerful tools for probabilistic inference. Sequential Monte Carlo (Moral, 1997) or particle filtering (Swendsen & Wang, 1986) has been the classical way to approximate complex posterior distributions over state-space models. Particle Gibbs (PG) sampling (Andrieu et al., 2010) extends these approaches by integrating MCMC techniques for improved inference.

3. Background

State space models are a class of probabilistic models used to describe sequential systems that evolve stepwise, typically over time (Särkkä, 2013). They consist of a sequence of hidden states $\{x_t\}_{t=1}^T$ and corresponding observations $\{o_t\}_{t=1}^T$, where $x_t \in \mathcal{X}$ represents the latent state at step t, and $o_t \in \mathcal{Y}$ is the observation. The evolution of states is governed by a transition model $p(x_t \mid x_{<t-1})$, and the observations are governed by the emission model $p(o_t|x_t)$. The joint distribution of states and observations is given by: $p(x_{1:T}, o_{1:T}) = p(x_1)\prod_{t=2}^T p(x_t \mid x_{<t-1})\prod_{t=1}^T p(o_t \mid x_t)$, where $p(x_1)$ is the prior distribution over the initial state.

Probabilistic inference in SSMs involves estimating the posterior distribution of the hidden states given the observations, $p(x_{1:T}|o_{1:T})$ (Särkkä, 2013). This task is generally intractable due to the high dimensionality of the state space and the dependencies in the model. Common approaches approximate the posterior through sampling-based methods or variational approaches (MacKay, 2003).

Particle filtering (PF) is a sequential Monte Carlo method to approximate the posterior distribution in SSMs (Swendsen & Wang, 1986; Moral, 1997). PF represents the posterior using a set of N weighted particles $\{x_t^{(i)}, w_t^{(i)}\}_{i=1}^N$, where $x_t^{(i)}$ denotes the i^{th} particle at time t, and $w_t^{(i)}$ is its associated weight. The algorithm iteratively propagates particles using the transition model and updates weights based on the emission model: $w_t^{(i)} \propto w_{t-1}^{(i)} p(o_t \mid x_t^{(i)})$.

4. Method

We begin by formulating inference-time scaling for LLMs as probabilistic inference over a state-space model (SSM), where the transition kernel is defined by the LLM and the emission probabilities are given by the PRM (Section 4.1). Next, in Section 4.2, we introduce how particle filtering (PF) can be applied to this inference task. We then extend our approach to incorporate multiple iterations and parallel chains, providing more ways to allocate computation budgets.

4.1. Inference-time scaling LLMs with PRMs as probabilistic inference over SSMs

For a LLM M (or p_M), our approach to inference-time scaling attempts to estimate the latent states of the following joint distribution over tokens (or chunks, e.g. steps in math problems) $x_{1:T}$ and observations $o_{1:T}$ representing the acceptance of the tokens, given prompt c

$$p_M(x_{1:T}, o_{1:T} \mid c) \propto \prod_{t=1}^T p_M(x_t \mid c, x_{<t-1}) \prod_{t=1}^T p(o_t \mid c, x_t), \text{ where} \quad (1)$$

- The transition kernel $p_M(x_t \mid c, x_{<t-1})$ is defined by M;
- The emission model or likelihood $p(o_t \mid c, x_t) = \mathcal{B}(o_t; r(c, x_t))$ is a Bernoulli whose parameter is defined by a reward function r of each x_t for prompt c.

Figure 1 shows the plate diagram of this SSM we define.

In inference-time scaling, we would like to find the sequence of latent states such that all steps are accepted ($o_t = 1$ for all t), i.e. estimating $p_M(x_{1:T} \mid c, o_{1:T} = \mathbf{1})$. This interpretation makes PF directly applicable.

Further, as the optimal or the ground-truth reward function r is often unknown in practice, we approximate r via a model that is suitable for the task. Following previous works, we use pre-trained PRMs $\hat{r}$ for such approximation when solving reasoning tasks in the domain of mathematics (for example), which gives us an approximate likelihood $\hat{p}(o_t \mid c, x_t) = \mathcal{B}(o_t; \hat{r}(c, x_t))$. Thus, our task is to estimate the latent states of the following joint given $o_t = 1$ for all t

$$\hat{p}_M(x_{1:T}, o_{1:T} \mid c) \propto \prod_{t=1}^{T} p_M(x_t \mid c, x_{<t-1}) \prod_{t=1}^{T} \hat{p}(o_t \mid c, x_t). \tag{2}$$

Sampling v.s. search An alternative to our sampling-based approach would be to find a point estimation of the distribution via optimization, which essentially reduces to variants of existing search-based inference-time scaling methods like MCTS, beam search, etc. However, we argue that such search-based methods are not robust in the case of PRM-based noisy approximations to the reward function. On the other hand, sampling using (2) can produce a closer estimation of (1) than optimization. This can be understood by comparing the typical set and the mode of a distribution: the mode of (2) is more sensitive to approximation errors in $\hat{r}$ than the typical set. This aligns with the classic insight that while sampling-based methods remain invariant to reparameterization in the likelihood, maximum-a-posteriori (MAP) inference—which underlies search-based methods—does not (Murphy, 2012).

In essence, sampling-based approaches are more robust to approximation errors in the likelihood, making them a better fit for this task—an advantage we will demonstrate empirically in the following sections.

4.2. Particle filtering for inference-time scaling

We now consider inference-time scaling with an LLM p_M and a PRM $\hat{r}$ via sampling from the posterior of (2) by conditioning on accepting all steps. The direct application of the classic particle filtering algorithm to this inference-time scaling setup requires defining the following components.

- State initialization and transition $p_M(x_t \mid c, x_{<t-1})$ is done by prompting the LLM with the prompt c to generate responses for a single step. These steps are determined automatically through stop-token delimiters. The LLM temperature is a hyperparameter to tune optimally for different tasks (see ablation in Section 5.4);
- Weight update $w_t^{(i)} \propto w_{t-1}^{(i)} \hat{r}(x_t^{(i)})$ uses the PRM to compute the reward per step, as detailed next.

PRMs for likelihood estimation and weight update How to aggregate the step-level rewards remains a choice when one uses PRMs. There are three common ways to assign rewards to a partial answer using PRMs: prod, which takes the product of rewards across all steps; $\min$, which selects the minimum reward over all steps; and last, which uses the reward from the final step. Zhang et al. (2025b) studies the optimal way for reward aggregation and points out that the "best choice" depends on if the PRM training data is prepared using MC rollout and/or human/model annotation. While prod aligns directly with the weight update rule described earlier, $\min$ and last do not allow for online weight updates. Therefore, for these methods, we compute the weight based on the entire partial trajectory instead.

Beyond these three approaches, we also explored a model-based reward aggregation method that performed surprisingly well. This method feeds the PRM with partial answers but only considers the final reward token, effectively prompting the model to provide an aggregated reward for the partial answer. Interestingly, we tested the Qwen PRM both for its original purpose as a true process reward model and repurposed as an outcome reward model. When used as a true PRM, it receives the question and a list of steps generated by the policy model, calculates scores for each step and selects the last score—a practice introduced and evaluated in Beeching et al. (2024). As an ORM, the PRM takes in a question and a concatenated string of generated steps, producing a score that we convert into a weight for the resampling process. Appendix A.2 provides an illustration of how the two input formats are structured. We compare various reward models and evaluate all four aggregation strategies through an ablation study in Section 5.4.

With the above defined, particle filtering iterates over the two steps below with a set of N particles at each iteration t

- Propagation: We start by propagating the set of particles S_{t-1} via initialization ($t = 1$) or transition ($t > 1$) and calculate their weights. This produces a set of weighted particles $\mathcal{S}'_t = \{x_t^{(i)}, w_t^{(i)}\}$, which represents partial generation upto step t and their importance;
- Resampling: We *sample with replacement* over the particles to produce a new set of particles with the same number. Specifically, let the resampling distribution (over index j) be

$$\mathbb{P}_t(j = i) = \exp(w_t^{(i)}) / \sum_{i'=1}^{n} \exp(w_t^{(i')}). \tag{3}$$

Algorithm 1 Particle Filtering for Inference-Time Scaling

Input: the number of particles N, a reward model $\hat{r}$, a LLM p_M and the prompt c
Initialize N particles $\{x_1^{(i)} \sim p_M(\cdot \mid c)\}_{i=1}^N$
$t \leftarrow 1$
while not all particles stop **do**
  Update rewards $\mathbf{w} = [\hat{r}(x_{1:t}^{(1)}), \ldots, \hat{r}(x_{1:t}^{(N)})]$
  Compute softmax distribution $\theta = \mathrm{softmax}(\mathbf{w})$
  Sample indices $\{j_t^{(i)}\}_{i=1}^N \sim \mathbb{P}_t(j = i) = \theta_i$
  Update the set of particles as $\{x_{1:t}^{(j_t^{(i)})}\}_{i=1}^N$
  Transition $\{x_{t+1}^{(i)} \sim p_M(\cdot \mid c, x_{1:t}^{(i)})\}_{i=1}^N$
  $t \leftarrow t + 1$
end while
Return: the set of particles in the end

We sample $\{j_t^{(i)} \sim \mathbb{P}_t(j = i)\}_{i=1}^M$ and obtain a new set of particles $\mathcal{S}_t = \{x_t^{j_t^{(i)}}, w_t^{j_t^{(i)}}\}$. This step is essentially a probabilistic search with higher chances to explore high reward partial generations: These weights do not blindly guide the selection of high-reward particles at every stage of the search—they retain a degree of stochasticity that *encourages exploration* of under-explored regions of the sample space—explorations that may discover higher value answers later on.

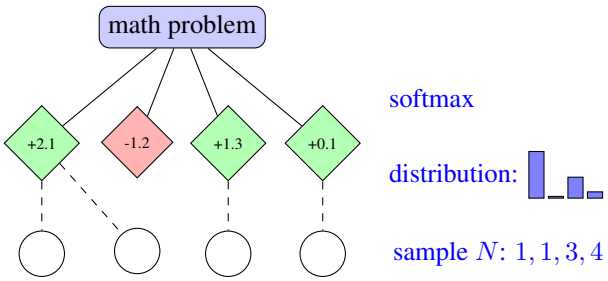
Note that the resampling step in particle filtering maintains a natural balance between exploiting promising hypotheses and exploring less-certain regions that may yield novel solutions. By maintaining a diverse population of particles and dynamically adjusting their weights at each step, our method allows a level of flexibility that is absent in traditional strategies, such as greedy search or beam search. In general, the ability to guide exploration using PRM-based scores allows the framework to harness the strengths of reward models without being limited by their flaws.

Importantly, this approach ensures that inference scaling remains fruitful within smaller compute budgets, as the resampling and unrolling operations are computationally efficient and can be parallelized across particles. With proper prefix caching, the total computation on generation is as much as that for generating N complete answers directly.

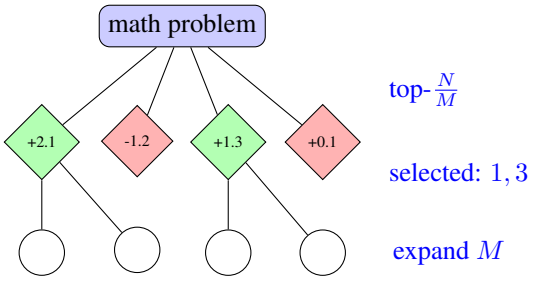
Figure 3 provides an illustration of the method with 4 particles with comparison to beam search, and the overall algorithm is detailed in Algorithm 1.

4.2.1. Multiple iterations and parallel chains

The PF approach to inference-time scaling can be used to define a MCMC kernel that enables two new types of scaling: multiple iterations of complete answers inspired by PG and parallel simulations inspired by parallel tempering.



(a) Particle filtering uses the rewards to produce a softmax distribution and does stochastic expansion of N based sampling.



(b) Beam search treats the rewards as exact and performs deterministic expansion based on beam size N and beam width M.

Figure 3. A side-by-side comparison between particle filtering and its closet search-based counterpart, beam search. Compared with beam search in Figure 3b where the selection and expansion is deterministic (implicitly assumes the rewards are correct), particle filtering in Figure 3a trust the rewards with uncertainty and propagate the expansion via sampling. A more detailed, step-by-step version of particle filtering can be found in Figure 9 of Appendix A.1.

Particle Gibbs is a type of MCMC algorithm that uses PF as a transition kernel (Andrieu et al., 2010). Specifically, at each iteration, PG samples a new set of particles using PF with a reference particle from the previous iteration. This integration combines the efficiency of PF with the theoretical guarantees of MCMC, making PG suitable for high-dimensional or challenging posterior distributions. The adaption of PG to inference-time scaling is essentially a multi-iteration extension of the PF algorithm presented, which works as follows: For each iteration, we run a modified PF step with an additional sampling step to sample 1 reference particle according to (3). For any PF step that is not the initial step, the PF is executed with a reference particle: This reference particle is never replaced during the resampling step, but its partial trajectory can still be forked during resampling. We detail the PG version of inference-time scaling in Algorithm 2 of Appendix A.1. Note that typically, a reasonably large number of particles is needed to show the benefits of multiple iterations, which we also confirm in our results in Section 5.4.

Parallel tempering In parallel tempering (aka replica exchange MCMC sampling), multiple MCMC chains run in parallel at different temperatures and swap the states to allow

better exploration. The key idea is that the chain running in high temperature can explore better, e.g. traversing between different modes of the target, and the swap makes it possible to let the low temperature chain exploit the new region found by the other chain. We detail the complete parallel tempering version of inference-time scaling in Algorithm 3 of Appendix A.1 while we only explore a special case of it (multiple chains with single iteration) in our experiments.

5. Evaluation

We thoroughly evaluate our proposed methods in this section. We detail our experimental setup in Section 5.1 and start with highlighted results on comparison with other closed-source models and competitive inference-time scaling methods with open-source models (Section 5.2). We then study how the main algorithm, particle filtering, scales with more computation and compare it with its competitors (Section 5.3). We further perform an extensive ablation study on key algorithmic choices like reward models, reward aggregation and LLM temperatures (Section 5.4). We finally study different possible allocations of the computation budget through iterative and parallel extensions (Section 5.5).

5.1. Setup

Models We consider two types of open-source small language models (SLMs) as our policy models for generating solutions. The first is general-purpose models, of which we used Llama-3.2-1B-Instruct and Llama-3.1-8B-Instruct (Grattafiori et al., 2024). The second is math-specialized models, where we used Qwen2.5-Math-1.5B-Instruct and Qwen2.5-Math-7B-Instruct (Yang et al., 2024). These small models are well-suited for inference-time scaling, enabling efficient exploration of multiple trajectories.

Process Reward Models To guide our policy models, we utilized Qwen2.5-Math-PRM-7B (Zhang et al., 2025a), a 7B process reward model. We selected this model because it demonstrated superior performance compared to other PRMs we tested, including Math-Shepherd-mistral-7b-prm (Wang et al., 2024), Llama3.1-8B-PRM-Deepseek-Data (Xiong et al., 2024), and EurusPRM-Stage2 (Yuan et al., 2024). This result as an ablation study is provided in Section 5.4, where we also study the different ways to aggregate step-level rewards from PRMs discussed in Section 4.2.

Baselines

- Pass@1: single greedy generation from the model, serving as the "bottom-line" performance.
- BoN/WBoN (Brown et al., 2024): (weighted) best-of-N is the most straightforward inference-time scaling method using reward models.
- DVTS (Beeching et al., 2024): a parallel extension of beam search that improves the exploration hence overall scaling performance.[1]

Datasets To evaluate our methods and baselines, we consider widely-used datasets spanning multiple domains and difficulty levels and challenging benchmarks, ensuring a robust assessment of the methods' performance across basic and advanced problem-solving and reasoning tasks.

- **MATH500** (Lightman et al., 2023b): A dataset containing 500 high-difficulty competition-level problems from various mathematical domains.
- **AIME 2024** (AI-MO, 2023): A collection of 30 problems from the American Invitational Mathematics Examination (AIME I and II) 2024.

Parsing and scoring To evaluate model-generated responses, we enforce a structured answer format using a system prompt (see Appendix A.2). This prompt ensures that the final answer is enclosed within a `\boxed{}` expression, facilitating automated extraction. We provide a detailed version of our scoring process in Appendix A.3.

5.2. Main results

We first present our main results, comparing our approach against a set of strong baselines in Table 1. Inference-time scaling results are based on a budget of 64 samples, with Qwen2.5-Math-PRM-7B serving as the reward model. Specifically, it is used as an ORM in WBoN and as a PRM otherwise.

- Among all inference-time scaling methods, **PF consistently achieves the best performance**, outperforming other scaling methods by a significant margin.
- **PF with Llama-3.1-8B-Instruct** outperforms its much larger counterpart, **Llama-3.1-70B-Instruct**, on **MATH500** and achieves parity on **AIME 2024**, demonstrating the efficiency of our approach.
- The **best PF results with Qwen2.5-Math-1.5B-Instruct** surpass **GPT-4o** on both datasets, while coming very close to the **o1-preview** model on the MATH500 benchmark while the Qwen2.5-Math-7B-Instruct is able to match the performance of **o1-preview** on MATH500, further underscoring the effectiveness of our method.

5.3. Scaling with inference-time compute

We now zoom in on how PF scales with inference-time compute. Figure 2 shows the change of performance (in terms of accuracy) with an increasing computation budget

[1]We only consider DVTS but not beam search itself for two reasons. First, it has been reported by Beeching et al. (2024) to have a better performance than beam search when the budget is more than 16. Second, the implementation of beam search from the official release by Beeching et al. (2024) is slower than DVTS on the same budget.

Model	Method	MATH500	AIME 2024
Closed-Source LLMs			
GPT-4o	-	76.2	13.3
o1-preview	-	***87.0***	***40.0***
Claude3.5-Sonnet	-	78.3	16.0
Open-Source LLMs			
Llama-3.1-70B-Instruct	-	65.7	16.6
Qwen2.5-Math-72B-Instruct	-	82.0	30.0
Open-Source SLMs			
Llama-3.2-1B-Instruct	Pass@1	26.8	0.0
	BoN	46.6	3.3
	WBoN	47.8	3.3
	DVTS	52.8	6.6
	Ours - PF	**59.6**	**10.0**
Llama-3.1-8B-Instruct	Pass@1	49.9	6.6
	BoN	58.6	10.0
	WBoN	59.0	10.0
	DVTS	65.7	13.3
	Ours - PF	**74.4**	**16.6**
Open-Source Math SLMs			
Qwen2.5-Math-1.5B-Instruct	Pass@1	70.0	10.0
	BoN	82.6	13.3
	WBoN	82.8	13.3
	DVTS	83.4	16.6
	Ours - PF	**85.4**	**23.3**
Qwen2.5-Math-7B-Instruct	Pass@1	79.6	16.6
	BoN	83.0	20.0
	WBoN	84.6	20.0
	DVTS	85.4	20.0
	Ours - PF	***87.0***	**23.3**

Table 1. Results of various LLMs on MATH500 and AIME 2024 where **bold** indicates the best in each category and *italic* indicates the overall best. The table highlights the performance of Inference Scaling methods, where Qwen2.5-Math-PRM-7B was used as the Reward Model. Each inference scaling methods were run with a computational budget of 64 model generations. Notably, the Qwen2.5-Math-7B model, when scaled with inference-time compute, achieves performance on par with o1-preview in MATH500, further showcasing the power of inference-time scaling for competitive performance with smaller models.

($N = 1, 2, 4, 8, 16, 32, 64, 128$) for all SLMs we consider. As we can see, PF scales 4–16x faster than the next best competitor DVTS, e.g. DVTS requires a budget of 32 to reach the same performance of PF with a budget of 8 with LLama-3.2-1B-Instruct and requires a budget of 128 to reach the performance of PF with a budget of 8 with LLama-3.1-8B-Instruct.

5.4. Ablation study

Performance of different PRMs To investigate the impact of the choice of PRM on our method, in Figure 4 we present the results of an ablation study on a subset of 100 questions from the MATH500 dataset, where we compare the accuracy of our method across various reward functions as the number of particles increases. Qwen2.5-Math-PRM-7B consistently outperforms other models, making it the natural choice for our main results. Interestingly, while EurusPRM-Stage2 performs relatively poorly with smaller budgets, it gradually improves and eventually matches Qwen2.5-Math-PRM-7B at higher budgets.

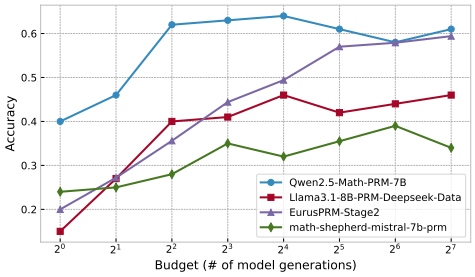


Figure 4. Results of ablation on 100 question subset comparing the performance of PF across various PRMs. We find that the Qwen PRM scales the most effectively across generations.

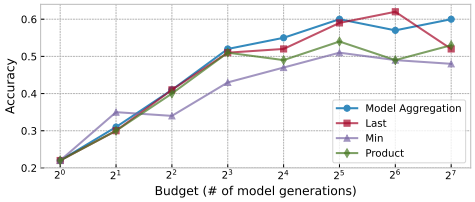


Figure 5. Effect of different aggregation strategies for the process reward model Qwen2.5-Math-PRM-7B, evaluated on a 100-question subset of the MATH500 dataset. The plot compares the commonly used aggregation strategies—Min, Last, and Product—against our proposed Model Aggregation method.

Reward aggregation within PRMs As mentioned in Section 4.2 and reported by many previous works (Zhang et al., 2025b), there exist multiple ways to use PRMs to calculate reward scores which can have large impact on final performance. Figure 5 studies 3 existing ways to use a set of PRM scores—using the *last* reward, the *minimum* reward, and the *product* of all the rewards. We also study "Model Aggregation", through which we use the PRM as an ORM with partial answers. As we can see, using Model Aggregation—in essence, feeding into a PRM the entire partial answer alongside the question - scales the best with an increasing budget.

Controlling the state transition—temperatures in LLM generation We investigate the effect of different LM sampling temperatures on the scaling of our method across different numbers of particles. The results of our ablation study on a 100 question subset of MATH questions are shown in Figure 6. Our findings indicate that the commonly used range of llm temperature of 0.4–1.0 performs well, with minimal variations in accuracy across different budgets. Similar to Beeching et al. (2024), we set the temperature to 0.8 for all our experiments.

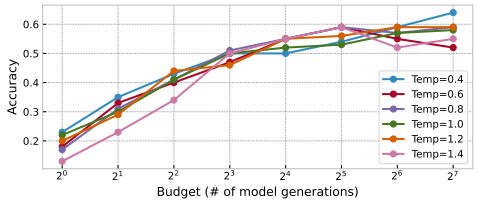


Figure 6. Results of using Llama 3.2 1B as our policy model across temperatures (0.4, 0.6, 0.8, 1.0, 1.2, and 1.4) and particle numbers (1, 2, 4, 8, 16, and 32).

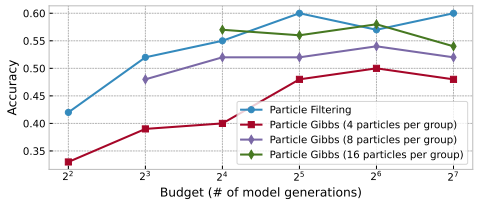


Figure 7. Comparison of PF and Particle Gibbs with different numbers of iterations, evaluated on a 100-question subset of the MATH-500 dataset using Llama-3.2-1B-Instruct as the policy model.

5.5. Budget allocation over iterations and parallelism

The multi-iteration and parallel-chain extensions introduced in Section 4.2.1 provides two more axes to spend computation in addition to the number of particles. We now explore how different ways to allocate budgets changes the performance. Specifically, we study for a fixed budget $N \times T \times M$, how the combination of N, T, M can yield the best performance, where N is the number of particles, T is the number of iterations, and M is the number of parallelism.

Allocating budget between N and T Figure 7 shows results of Llama-3.2 1B model when configured with various test-time compute budget allocations. Although the plot shows that various Particle Gibbs configurations do not have a marked benefit over an equivalently budgeted particle filtering run, a PG experiment with 16 particles and 4 iterations powered by a Qwen 2.5 7B Math Instruct policy model achieved a 87.2% accuracy on MATH500, beating o1 performance. Configurations with larger N values typically do better than equivalently budgeted runs with less particles.

Allocating budget between N and M Figure 8 shows PF and 3 PT configurations over a set of increasing numbers of budgets. First, as we can see, for any fixed N, increasing M also improves the performance. This may be helpful when combining batch generation with distributed computing. Second, PT with $N = 16$ has a better overall scaling than PF. This indicates that there is some optimal budget allocation over parallel chains that can further improve the overall performance of our main results.

We leave the exploration over the optimal configuration of N, T, M jointly as a future work.

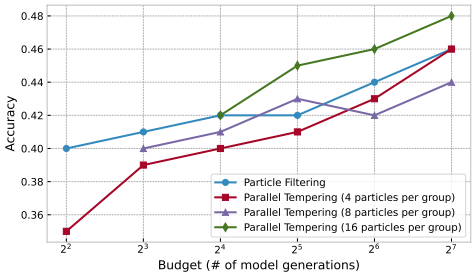


Figure 8. Comparison of PF and PT with different particle group sizes, evaluated on a 100-question subset of the MATH500 dataset using Llama-3.2-1B-Instruct as the policy model.

6. Conclusion

In this paper, we introduce a set of inference-time scaling algorithms with PRMs that leverage particle-based Monte Carlo methods. Our evaluation demonstrates that these algorithms consistently outperform search-based approaches by a significant margin.

However, inference-time scaling comes with computational challenges. Hosting and running a reward model often introduces high latency, making the process more resource-intensive. Additionally, for smaller models, extensive prompt engineering is often required to ensure outputs adhere to the desired format. Finally, hyperparameters such as temperature are problem-dependent and may require extensive tuning across different domains.

We hope that the formal connection of inference scaling to probabilistic modeling that we established in this work will lead to systematic solutions for the current limitations of these methods and pave the way for bringing advanced probabilistic inference algorithms into LLM inference-time scaling in future work.

References

AI-MO. Aimo validation aime dataset. https://huggingface.co/datasets/AI-MO/aimo-validation-aime, 2023. Accessed: 2025-01-24.

Andrieu, C., Doucet, A., and Holenstein, R. Particle Markov Chain Monte Carlo Methods. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 72(3): 269–342, June 2010. ISSN 1369-7412, 1467-9868. doi: 10.1111/j.1467-9868.2009.00736.x.

Beeching, E., Tunstall, L., and Rush, S. Scaling test-time compute with open models, 2024. URL https://huggingface.co/spaces/HuggingFaceH4/blogpost-scaling-test-time-compute.

Brown, B., Juravsky, J., Ehrlich, R., Clark, R., Le, Q. V., Ré, C., and Mirhoseini, A. Large Language Monkeys: Scaling Inference Compute with Repeated Sampling, July 2024.

Cui, G., Yuan, L., Wang, Z., Wang, H., Li, W., He, B., Fan, Y., Yu, T., Xu, Q., Chen, W., Yuan, J., Chen, H., Zhang, K., Lv, X., Wang, S., Yao, Y., Peng, H., Cheng, Y., Liu, Z., Sun, M., Zhou, B., and Ding, N. Process reinforcement through implicit rewards, 2025.

Grattafiori, A., Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Vaughan, A., Yang, A., Fan, A., Goyal, A., Hartshorn, A., Yang, A., Mitra, A., Sravankumar, A., Korenev, A., Hinsvark, A., Rao, A., Zhang, A., Rodriguez, A., Gregerson, A., Spataru, A., Roziere, B., Biron, B., Tang, B., Chern, B., Caucheteux, C., Nayak, C., Bi, C., Marra, C., McConnell, C., Keller, C., Touret, C., Wu, C., Wong, C., Ferrer, C. C., Nikolaidis, C., Allonsius, D., Song, D., Pintz, D., Livshits, D., Wyatt, D., Esiobu, D., Choudhary, D., Mahajan, D., Garcia-Olano, D., Perino, D., Hupkes, D., Lakomkin, E., AlBadawy, E., Lobanova, E., Dinan, E., Smith, E. M., Radenovic, F., Guzmán, F., Zhang, F., Synnaeve, G., Lee, G., Anderson, G. L., Thattai, G., Nail, G., Mialon, G., Pang, G., Cucurell, G., Nguyen, H., Korevaar, H., Xu, H., Touvron, H., Zarov, I., Ibarra, I. A., Kloumann, I., Misra, I., Evtimov, I., Zhang, J., Copet, J., Lee, J., Geffert, J., Vranes, J., Park, J., Mahadeokar, J., Shah, J., van der Linde, J., Billock, J., Hong, J., Lee, J., Fu, J., Chi, J., Huang, J., Liu, J., Wang, J., Yu, J., Bitton, J., Spisak, J., Park, J., Rocca, J., Johnstun, J., Saxe, J., Jia, J., Alwala, K. V., Prasad, K., Upasani, K., Plawiak, K., Li, K., Heafield, K., Stone, K., El-Arini, K., Iyer, K., Malik, K., Chiu, K., Bhalla, K., Lakhotia, K., Rantala-Yeary, L., van der Maaten, L., Chen, L., Tan, L., Jenkins, L., Martin, L., Madaan, L., Malo, L., Blecher, L., Landzaat, L., de Oliveira, L., Muzzi, M., Pasupuleti, M., Singh, M., Paluri, M., Kardas, M., Tsimpoukelli, M., Oldham, M., Rita, M., Pavlova, M., Kambadur, M., Lewis, M., Si, M., Singh, M. K., Hassan, M., Goyal, N., Torabi, N., Bashlykov, N., Bogoychev, N., Chatterji, N., Zhang, N., Duchenne, O., Çelebi, O., Alrassy, P., Zhang, P., Li, P., Vasic, P., Weng, P., Bhargava, P., Dubal, P., Krishnan, P., Koura, P. S., Xu, P., He, Q., Dong, Q., Srinivasan, R., Ganapathy, R., Calderer, R., Cabral, R. S., Stojnic, R., Raileanu, R., Maheswari, R., Girdhar, R., Patel, R., Sauvestre, R., Polidoro, R., Sumbaly, R., Taylor, R., Silva, R., Hou, R., Wang, R., Hosseini, S., Chennabasappa, S., Singh, S., Bell, S., Kim, S. S., Edunov, S., Nie, S., Narang, S., Raparthy, S., Shen, S., Wan, S., Bhosale, S., Zhang, S., Vandenhende, S., Batra, S., Whitman, S., Sootla, S., Collot, S., Gururangan, S., Borodinsky, S., Herman, T., Fowler, T., Sheasha, T., Georgiou, T., Scialom, T., Speckbacher, T., Mihaylov, T., Xiao, T., Karn, U., Goswami, V., Gupta, V., Ramanathan, V., Kerkez, V., Gonguet, V., Do, V., Vogeti, V., Albiero, V., Petrovic, V., Chu, W., Xiong, W., Fu, W., Meers, W., Martinet, X., Wang, X., Wang, X., Tan, X. E., Xia, X., Xie, X., Jia, X., Wang, X., Goldschlag, Y., Gaur, Y., Babaei, Y., Wen, Y., Song, Y., Zhang, Y., Li, Y., Mao, Y., Coudert, Z. D., Yan, Z., Chen, Z., Papakipos, Z., Singh, A., Srivastava, A., Jain, A., Kelsey, A., Shajnfeld, A., Gangidi, A., Victoria, A., Goldstand, A., Menon, A., Sharma, A., Boesenberg, A., Baevski, A., Feinstein, A., Kallet, A., Sangani, A., Teo, A., Yunus, A., Lupu, A., Alvarado, A., Caples, A., Gu, A., Ho, A., Poulton, A., Ryan, A., Ramchandani, A., Dong, A., Franco, A., Goyal, A., Saraf, A., Chowdhury, A., Gabriel, A., Bharambe, A., Eisenman, A., Yazdan, A., James, B., Maurer, B., Leonhardi, B., Huang, B., Loyd, B., Paola, B. D., Paranjape, B., Liu, B., Wu, B., Ni, B., Hancock, B., Wasti, B., Spence, B., Stojkovic, B., Gamido, B., Montalvo, B., Parker, C., Burton, C., Mejia, C., Liu, C., Wang, C., Kim, C., Zhou, C., Hu, C., Chu, C.-H., Cai, C., Tindal, C., Feichtenhofer, C., Gao, C., Civin, D., Beaty, D., Kreymer, D., Li, D., Adkins, D., Xu, D., Testuggine, D., David, D., Parikh, D., Liskovich, D., Foss, D., Wang, D., Le, D., Holland, D., Dowling, E., Jamil, E., Montgomery, E., Presani, E., Hahn, E., Wood, E., Le, E.-T., Brinkman, E., Arcaute, E., Dunbar, E., Smothers, E., Sun, F., Kreuk, F., Tian, F., Kokkinos, F., Ozgenel, F., Caggioni, F., Kanayet, F., Seide, F., Florez, G. M., Schwarz, G., Badeer, G., Swee, G., Halpern, G., Herman, G., Sizov, G., Guangyi, Zhang, Lakshminarayanan, G., Inan, H., Shojanazeri, H., Zou, H., Wang, H., Zha, H., Habeeb, H., Rudolph, H., Suk, H., Aspegren, H., Goldman, H., Zhan, H., Damlaj, I., Molybog, I., Tufanov, I., Leontiadis, I., Veliche, I.-E., Gat, I., Weissman, J., Geboski, J., Kohli, J., Lam, J., Asher, J., Gaya, J.-B., Marcus, J., Tang, J., Chan, J., Zhen, J., Reizenstein, J., Teboul, J., Zhong, J., Jin, J., Yang, J., Cummings, J., Carvill, J., Shepard, J., McPhie, J., Torres, J., Ginsburg, J., Wang, J., Wu, K., U,

K. H., Saxena, K., Khandelwal, K., Zand, K., Matosich, K., Veeraraghavan, K., Michelena, K., Li, K., Jagadeesh, K., Huang, K., Chawla, K., Huang, K., Chen, L., Garg, L., A, L., Silva, L., Bell, L., Zhang, L., Guo, L., Yu, L., Moshkovich, L., Wehrstedt, L., Khabsa, M., Avalani, M., Bhatt, M., Mankus, M., Hasson, M., Lennie, M., Reso, M., Groshev, M., Naumov, M., Lathi, M., Keneally, M., Liu, M., Seltzer, M. L., Valko, M., Restrepo, M., Patel, M., Vyatskov, M., Samvelyan, M., Clark, M., Macey, M., Wang, M., Hermoso, M. J., Metanat, M., Rastegari, M., Bansal, M., Santhanam, N., Parks, N., White, N., Bawa, N., Singhal, N., Egebo, N., Usunier, N., Mehta, N., Laptev, N. P., Dong, N., Cheng, N., Chernoguz, O., Hart, O., Salpekar, O., Kalinli, O., Kent, P., Parekh, P., Saab, P., Balaji, P., Rittner, P., Bontrager, P., Roux, P., Dollar, P., Zvyagina, P., Ratanchandani, P., Yuvraj, P., Liang, Q., Alao, R., Rodriguez, R., Ayub, R., Murthy, R., Nayani, R., Mitra, R., Parthasarathy, R., Li, R., Hogan, R., Battey, R., Wang, R., Howes, R., Rinott, R., Mehta, S., Siby, S., Bondu, S. J., Datta, S., Chugh, S., Hunt, S., Dhillon, S., Sidorov, S., Pan, S., Mahajan, S., Verma, S., Yamamoto, S., Ramaswamy, S., Lindsay, S., Lindsay, S., Feng, S., Lin, S., Zha, S. C., Patil, S., Shankar, S., Zhang, S., Zhang, S., Wang, S., Agarwal, S., Sajuyigbe, S., Chintala, S., Max, S., Chen, S., Kehoe, S., Satterfield, S., Govindaprasad, S., Gupta, S., Deng, S., Cho, S., Virk, S., Subramanian, S., Choudhury, S., Goldman, S., Remez, T., Glaser, T., Best, T., Koehler, T., Robinson, T., Li, T., Zhang, T., Matthews, T., Chou, T., Shaked, T., Vontimitta, V., Ajayi, V., Montanez, V., Mohan, V., Kumar, V. S., Mangla, V., Ionescu, V., Poenaru, V., Mihailescu, V. T., Ivanov, V., Li, W., Wang, W., Jiang, W., Bouaziz, W., Constable, W., Tang, X., Wu, X., Wang, X., Wu, X., Gao, X., Kleinman, Y., Chen, Y., Hu, Y., Jia, Y., Qi, Y., Li, Y., Zhang, Y., Zhang, Y., Adi, Y., Nam, Y., Yu, Wang, Zhao, Y., Hao, Y., Qian, Y., Li, Y., He, Y., Rait, Z., DeVito, Z., Rosnbrick, Z., Wen, Z., Yang, Z., Zhao, Z., and Ma, Z. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Guan, X., Zhang, L. L., Liu, Y., Shang, N., Sun, Y., Zhu, Y., Yang, F., and Yang, M. rStar-Math: Small LLMs Can Master Math Reasoning with Self-Evolved Deep Thinking, January 2025.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models, 2020. URL https://arxiv.org/abs/2001.08361.

Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., Leike, J., Schulman, J., Sutskever, I., and Cobbe, K. Let's Verify Step by Step, May 2023a.

Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., Leike, J., Schulman, J., Sutskever, I., and Cobbe, K. Let's verify step by step, 2023b. URL https://arxiv.org/abs/2305.20050.

MacKay, D. J. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

Moral, P. D. Sequential Monte Carlo Methods for Dynamic Systems: Journal of the American Statistical Association: Vol 93, No 443. https://www.tandfonline.com/doi/abs/10.1080/01621459.1998.10473765, 1997.

Murphy, K. P. *Machine learning: a probabilistic perspective*. MIT press, 2012.

OpenAI, :, Jaech, A., Kalai, A., Lerer, A., Richardson, A., El-Kishky, A., Low, A., Helyar, A., Madry, A., Beutel, A., Carney, A., Iftimie, A., Karpenko, A., Passos, A. T., Neitz, A., Prokofiev, A., Wei, A., Tam, A., Bennett, A., Kumar, A., Saraiva, A., Vallone, A., Duberstein, A., Kondrich, A., Mishchenko, A., Applebaum, A., Jiang, A., Nair, A., Zoph, B., Ghorbani, B., Rossen, B., Sokolowsky, B., Barak, B., McGrew, B., Minaiev, B., Hao, B., Baker, B., Houghton, B., McKinzie, B., Eastman, B., Lugaresi, C., Bassin, C., Hudson, C., Li, C. M., de Bourcy, C., Voss, C., Shen, C., Zhang, C., Koch, C., Orsinger, C., Hesse, C., Fischer, C., Chan, C., Roberts, D., Kappler, D., Levy, D., Selsam, D., Dohan, D., Farhi, D., Mely, D., Robinson, D., Tsipras, D., Li, D., Oprica, D., Freeman, E., Zhang, E., Wong, E., Proehl, E., Cheung, E., Mitchell, E., Wallace, E., Ritter, E., Mays, E., Wang, F., Such, F. P., Raso, F., Leoni, F., Tsimpourlas, F., Song, F., von Lohmann, F., Sulit, F., Salmon, G., Parascandolo, G., Chabot, G., Zhao, G., Brockman, G., Leclerc, G., Salman, H., Bao, H., Sheng, H., Andrin, H., Bagherinezhad, H., Ren, H., Lightman, H., Chung, H. W., Kivlichan, I., O'Connell, I., Osband, I., Gilaberte, I. C., Akkaya, I., Kostrikov, I., Sutskever, I., Kofman, I., Pachocki, J., Lennon, J., Wei, J., Harb, J., Twore, J., Feng, J., Yu, J., Weng, J., Tang, J., Yu, J., Candela, J. Q., Palermo, J., Parish, J., Heidecke, J., Hallman, J., Rizzo, J., Gordon, J., Uesato, J., Ward, J., Huizinga, J., Wang, J., Chen, K., Xiao, K., Singhal, K., Nguyen, K., Cobbe, K., Shi, K., Wood, K., Rimbach, K., Gu-Lemberg, K., Liu, K., Lu, K., Stone, K., Yu, K., Ahmad, L., Yang, L., Liu, L., Maksin, L., Ho, L., Fedus, L., Weng, L., Li, L., McCallum, L., Held, L., Kuhn, L., Kondraciuk, L., Kaiser, L., Metz, L., Boyd, M., Trebacz, M., Joglekar, M., Chen, M., Tintor, M., Meyer, M., Jones, M., Kaufer, M., Schwarzer, M., Shah, M., Yatbaz, M., Guan, M. Y., Xu, M., Yan, M., Glaese, M., Chen, M., Lampe, M., Malek, M., Wang, M., Fradin, M., McClay, M., Pavlov, M., Wang, M., Wang, M., Murati, M., Bavarian, M., Rohaninejad, M., McAleese, N., Chowdhury, N., Chowdhury, N., Ryder, N., Tezak, N., Brown, N., Nachum, O., Boiko, O., Murk, O., Watkins, O., Chao, P.,

Ashbourne, P., Izmailov, P., Zhokhov, P., Dias, R., Arora, R., Lin, R., Lopes, R. G., Gaon, R., Miyara, R., Leike, R., Hwang, R., Garg, R., Brown, R., James, R., Shu, R., Cheu, R., Greene, R., Jain, S., Altman, S., Toizer, S., Toyer, S., Miserendino, S., Agarwal, S., Hernandez, S., Baker, S., McKinney, S., Yan, S., Zhao, S., Hu, S., Santurkar, S., Chaudhuri, S. R., Zhang, S., Fu, S., Papay, S., Lin, S., Balaji, S., Sanjeev, S., Sidor, S., Broda, T., Clark, A., Wang, T., Gordon, T., Sanders, T., Patwardhan, T., Sottiaux, T., Degry, T., Dimson, T., Zheng, T., Garipov, T., Stasi, T., Bansal, T., Creech, T., Peterson, T., Eloundou, T., Qi, V., Kosaraju, V., Monaco, V., Pong, V., Fomenko, V., Zheng, W., Zhou, W., McCabe, W., Zaremba, W., Dubois, Y., Lu, Y., Chen, Y., Cha, Y., Bai, Y., He, Y., Zhang, Y., Wang, Y., Shao, Z., and Li, Z. Openai o1 system card, 2024. URL https://arxiv.org/abs/2412.16720.

Pareja, A., Nayak, N. S., Wang, H., Killamsetty, K., Sudalairaj, S., Zhao, W., Han, S., Bhandwaldar, A., Xu, G., Xu, K., Han, L., Inglis, L., and Srivastava, A. Unveiling the secret recipe: A guide for supervised fine-tuning small llms, 2024. URL https://arxiv.org/abs/2412.13337.

Snell, C., Lee, J., Xu, K., and Kumar, A. Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters, August 2024.

Sudalairaj, S., Bhandwaldar, A., Pareja, A., Xu, K., Cox, D. D., and Srivastava, A. Lab: Large-scale alignment for chatbots, 2024. URL https://arxiv.org/abs/2403.01081.

Swendsen, R. H. and Wang, J.-S. Nonlinear filtering: Interacting particle resolution - ScienceDirect. https://www.sciencedirect.com/science/article/abs/pii/S0764444297847787, 1986.

Särkkä, S. *Bayesian Filtering and Smoothing*. Institute of Mathematical Statistics Textbooks. Cambridge University Press, 2013.

Wang, P., Li, L., Shao, Z., Xu, R. X., Dai, D., Li, Y., Chen, D., Wu, Y., and Sui, Z. Math-shepherd: Verify and reinforce llms step-by-step without human annotations, 2024. URL https://arxiv.org/abs/2312.08935.

Xiong, W., Zhang, H., Jiang, N., and Zhang, T. An implementation of generative prm. https://github.com/RLHFlow/RLHF-Reward-Modeling, 2024.

Yang, A., Zhang, B., Hui, B., Gao, B., Yu, B., Li, C., Liu, D., Tu, J., Zhou, J., Lin, J., Lu, K., Xue, M., Lin, R., Liu, T., Ren, X., and Zhang, Z. Qwen2.5-math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*, 2024.

Yuan, L., Cui, G., Wang, H., Ding, N., Wang, X., Deng, J., Shan, B., Chen, H., Xie, R., Lin, Y., Liu, Z., Zhou, B., Peng, H., Liu, Z., and Sun, M. Advancing llm reasoning generalists with preference trees, 2024. URL https://arxiv.org/abs/2404.02078.

Zhang, Z., Zheng, C., Wu, Y., Zhang, B., Lin, R., Yu, B., Liu, D., Zhou, J., and Lin, J. The lessons of developing process reward models in mathematical reasoning. *arXiv preprint arXiv:2501.07301*, 2025a.

Zhang, Z., Zheng, C., Wu, Y., Zhang, B., Lin, R., Yu, B., Liu, D., Zhou, J., and Lin, J. The Lessons of Developing Process Reward Models in Mathematical Reasoning, January 2025b.

Zhou, A., Yan, K., Shlapentokh-Rothman, M., Wang, H., and Wang, Y.-X. Language agent tree search unifies reasoning acting and planning in language models, 2024. URL https://arxiv.org/abs/2310.04406.

[h]

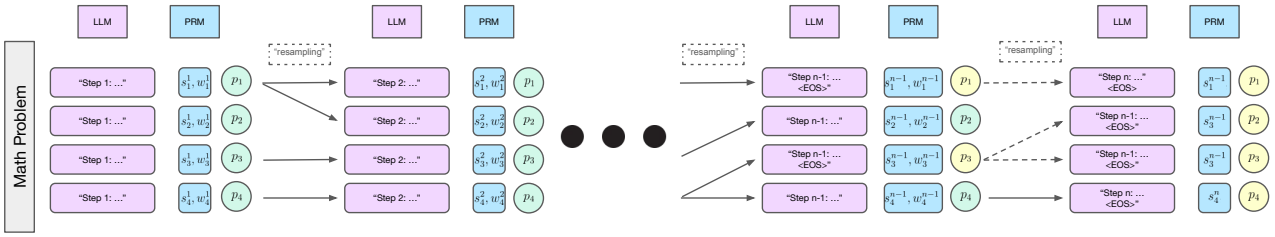


Figure 9. Particle filtering for inference scaling in details. We initialize x particles with the "first step" of an answer to a question. At every step, each particle p_i is given a score s_i^t by the PRM, which is then used as a weight w_i^t to determine how likely that particle is to be resampled (evolved via a solid line) at the next step. A particle is deemed "active" (green, in this diagram) until it generates an ⟨EOS⟩ token, after which it is still able to be resampled (evolved via a dashed line) but is not evolved further. This process continues until all particles have completed their answers and become inactive (filled yellow).

Algorithm 2 Particle Gibbs for Inference-Time Scaling

Input: same as Algorithm 1 with the number of Gibbs iterations T
Run Algorithm 1 to get a set of particles $\{x_{1:t}^{(i)}\}_{i=1}^N$
for $j = 1, \ldots, T$ **do**
  Compute rewards $\mathbf{w} = [\hat{r}(x_{1:t}^{(1)}), \ldots, \hat{r}(x_{1:t}^{(N)})]$
  Compute softmax distribution $\theta = \mathrm{softmax}(\mathbf{w})$
  Sample reference particle $x_{1:t}^{\text{ref}} := x_{1:t}^{(j)}$ where $j \sim \mathbb{P}(j = i) = \theta_i$
  Initialize $N-1$ particles $\{x_1^{(i)} \sim p_M(\cdot \mid c)\}_{i=1}^{N-1}$
  $t \leftarrow 1$
  while not all particles stop **do**
    Update $\mathbf{w} = [\hat{r}(x_{1:t}^{(1)}), \ldots, \hat{r}(x_{1:t}^{(N-1)}), \hat{r}(x_{1:t}^{\text{ref}})]$
    Compute softmax distribution $\theta = \mathrm{softmax}(\mathbf{w})$
    Sample indices $\{j_t^{(i)}\}_{i=1}^N \sim \mathbb{P}_t(j = i) = \theta_i$
    Update the set of particles as $\{x_{1:t}^{(j_t^{(i)})}\}_{i=1}^N$
    Transition $\{x_{t+1}^{(i)} \sim p_M(\cdot \mid c, x_{t+1}^{(i)})\}_{i=1}^N$
    $t \leftarrow t + 1$
  end while
end for
Return: the set of particles in the end

A. Appendix

A.1. Algorithm details

For a set of parallel chains with temperatures $T_1 > T_2 > \ldots$, at each iteration, we swap the states of every pair of neighboring chains $k, k+1$ with the following probability

$$A = \min\left(1, \frac{\pi_k(x^{(k+1)})\pi_{k+1}(x^{(k)})}{\pi_k(x^{(k)})\pi_{k+1}(x^{(k+1)})}\right), \tag{4}$$

where π_k, π_{k+1} are the two targets (with different temperatures) and x_k, x_{k+1} are their states before swapping.

Algorithm 3 Particle Gibbs with Parallel Tempering for Inference-Time Scaling

Input: same as Algorithm 2 with the number of parallel chains M and a list of temperature $T_1, \ldots, T_M$
for $j = 1, \ldots, T$ **do**
  for $k = 1, \ldots, M$ **do**
    if $j = 1$ **then**
      Run Algorithm 1 to get a set of particles $\{x_{1:t}^{(i)}\}_{i=1}^{N}$ for chain k
    else
      Initialize $N-1$ particles $\{x_1^{(i)} \sim p_M(\cdot \mid c)\}_{i=1}^{N-1}$
      $t \leftarrow 1$
      while not all particles stop **do**
        Update $\mathbf{w} = [\hat{r}(x_{1:t}^{(1)}), \ldots, \hat{r}(x_{1:t}^{(N-1)}), \hat{r}(x_{1:t}^{\text{ref}})]$
        Compute softmax distribution $\theta = \text{softmax}(\mathbf{w}/T_k)$
        Sample indices $\{j_t^{(i)}\}_{i=1}^{N} \sim \mathbb{P}_t(j = i) = \theta_i$
        Update the set of particles as $\{x_{1:t}^{(j_t^{(i)})}\}_{i=1}^{N}$
        Transition $\{x_{t+1}^{(i)} \sim p_M(\cdot \mid c, x_{t+1}^{(i)})\}_{i=1}^{N}$
        $t \leftarrow t + 1$
      end while
    end if
    Compute rewards $\mathbf{w} = [\hat{r}(x_{1:t}^{(1)}), \ldots, \hat{r}(x_{1:t}^{(N)})]$
    Compute softmax distribution $\theta = \text{softmax}(\mathbf{w}/T_k)$
    Sample reference particle $x_{1:t}^{\text{ref}} := x_{1:t}^{(j)}$ where $j \sim \mathbb{P}(j = i) = \theta_i$
  end for
  for $k = 1, \ldots, M-1$ **do**
    Exchange the reference particle between chain k and $k+1$ with probability according to (4)
  end for
end for
Return: M set of particles in the end

A.2. Inference Prompt Template

Evaluation System Prompt

```
Solve the following math problem efficiently and clearly:
    - For simple problems (2 steps or fewer):
    Provide a concise solution with minimal explanation.

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

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

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

Regardless of the approach, always conclude with:

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

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

PRM Input Format

```
## Step 1: [Concise description]
[Brief explanation and calculations]
<reward_token>
## Step 2: [Concise description]
[Brief explanation and calculations]
<reward_token>
## Step 3: [Concise description]
[Brief explanation and calculations]
<reward_token>
```

ORM Input Format

```
## Step 1: [Concise description]
[Brief explanation and calculations]
## Step 2: [Concise description]
[Brief explanation and calculations]
## Step 3: [Concise description]
[Brief explanation and calculations]
<reward_token>
```

A.3. Evaluation details

Parsing and scoring Following prior work on mathematical reasoning benchmarks (Yang et al., 2024), we apply their heuristic-based parsing and cleaning techniques to robustly extract the boxed expression. These heuristics account for variations in spacing, formatting inconsistencies, and other common artifacts observed in model outputs. For answer verification, we follow Beeching et al. (2024) and convert responses to canonical form. Ground truth and generated answers are transformed from LaTeX into SymPy expressions, simplified for normalization, and converted back to LaTeX. Exact match is determined using two criteria: numerical equality, where both expressions evaluate to the same floating-point value, and symbolic equality, where both are algebraically equivalent as SymPy expressions (Beeching et al., 2024). Accuracy is then computed as the fraction of problems where the generated answer exactly matches the ground truth.