

STEP-BY-STEP REASONING FOR MATH PROBLEMS VIA TWISTED SEQUENTIAL MONTE CARLO

Anonymous authors

Paper under double-blind review

ABSTRACT

Augmenting the multi-step reasoning abilities of Large Language Models (LLMs) has been a persistent challenge. Recently, verification has shown promise in improving solution consistency by evaluating generated outputs. However, current verification approaches suffer from sampling inefficiencies, requiring a large number of samples to achieve satisfactory performance. Additionally, training an effective verifier often depends on extensive process supervision, which is costly to acquire. In this paper, we address these limitations by introducing a novel verification method based on Twisted Sequential Monte Carlo (TSMC). TSMC sequentially refines its sampling effort to focus exploration on promising candidates, resulting in more efficient generation of high-quality solutions. We apply TSMC to LLMs by estimating the expected future rewards at partial solutions. This approach results in a more straightforward training target that eliminates the need for step-wise human annotations. We empirically demonstrate the advantages of our method across multiple math benchmarks, and also validate our theoretical analysis of both our approach and existing verification methods.

1 INTRODUCTION

In recent years, Large Language Models (LLMs) have achieved significant breakthroughs across various domains (Park et al., 2023; Kaddour et al., 2023; Song et al.; Li et al., 2023a; Wang et al., 2023a; Chen et al., 2023; Zheng et al., 2023; Wang et al., 2023c). However, their performance in multi-step reasoning tasks, such as solving complex mathematical or coding problems, remains notably constrained (Lightman et al., 2024; Huang et al., 2023). A key challenge arises from the high sensitivity of these tasks to individual errors at each step of reasoning. Autoregressive LLMs, in particular, struggle with maintaining consistency throughout the reasoning process, leading to solutions that are prone to mistakes or logical inconsistencies (Shen et al., 2021; Cobbe et al., 2021).

Verification (Cobbe et al., 2021; Uesato et al., 2022; Lightman et al., 2024) has emerged as an effective strategy to mitigate these issues. In a typical verification process, multiple solutions are sampled from the generator (the LLM), and an external verifier evaluates each of these solutions. The verification outcomes are then used to adjust the weight of each solution in determining the final answer. Since verification is generally simpler than generation, it tends to achieve higher accuracy and consistency compared to the methods that rely solely on the generator, such as majority voting (Wang et al., 2023e). There are two primary types of verifiers: the Outcome Reward Model (ORM) (Cobbe et al., 2021) and the Process Reward Model (PRM) (Uesato et al., 2022). The ORM evaluates the fully generated solution with a single scalar output representing the confidence score, and its training is straightforward, using outcome supervision based on comparing generated answers with ground truth. In contrast, the PRM focuses on providing rewards at each step of the reasoning process, giving more detailed feedback on the intermediate steps. Although empirical evidence suggests that PRM outperforms ORM (Lightman et al., 2024), there exists no simple metric in evaluating the correctness of each step and efficiently collecting process supervision for such intermediate steps remains a great challenge.

Despite being promising, existing verification methods are still limited in the following two areas:

- **(Problem I) Low sampling efficiency:** Current verification methods only evaluate fully generated solutions, without refining their quality during the generation process. Sampling efforts would be

wasted on partial solutions that are clearly incorrect. As a result, a large number of samples are needed to obtain even one correct solution, making the process inefficient and resource-intensive.

- **(Problem II) Difficulty in obtaining process supervision:** Training powerful verifiers like the PRM requires detailed step-wise supervision. Existing approaches either rely on human effort (Uesato et al., 2022; Lightman et al., 2024) or tree search (Wang et al., 2023d; Luo et al., 2024) for intermediate step annotations. However, both approaches are inefficient and lack scalability, limiting their practical application for large-scale tasks.

To address these two significant problems in existing verification methods, we propose a novel approach based on *Twisted Sequential Monte Carlo (TSMC)* (Doucet et al., 2001; Del Moral et al., 2006; Briers et al., 2009; Chopin & Papaspiliopoulos, 2020). TSMC is a significant advancement in the *Importance Sampling (IS)* technology. Building on the foundation of Sequential Monte Carlo (SMC), TSMC is intended to enhance sampling efficiency of IS in the high-dimensional space. It employs a series of intermediate target distributions at each resampling step, which are defined through twist functions. This function strategically guides the samples towards the high-density region in the target distribution. By retaining the most promising samples, TSMC effectively reduces the variance in estimated quantities and boosts the efficiency of the sampling process.

Notably, the application of TSMC to improve the verification method in LLMs has not been explored previously, making our study the first of its kind in this area. Our approach is inspired by the realization that existing verification methods employing reward-weighted majority voting (Li et al., 2023b) essentially performs IS, where the sampling efficiency deteriorates as the disparity between the proposal distribution (which generates potential solutions) and the target distribution (concentrated around correct solutions) widens. We identify Problem I—low sampling efficiency—as a consequence of high variance in IS when there is a substantial deviation between the proposal and target distributions. Multi-step reasoning, even minor discrepancies at each step can cumulate into a substantial mismatch between the two distributions. We therefore apply TSMC to improve the sampling efficiency of verification by focusing the sampling effort on promising partial solutions during the intermediate decoding process. We have shown the optimal twist functions in our case, which is used to guide the sampling of TSMC, is proportional to expected future rewards, also known as the value function. The value function could be simply learnt through a neural regressor on the data independently sampled from the generator. This simplifies the training target by eliminating the need for human annotations or tree search. We also highlight the relationship between TSMC and the PRM in existing verification methods, allowing for a comprehensive analysis of bias and variance.

We compare our proposed method with baseline approaches on two math benchmarks: GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021), utilizing fine-tuned models from Llemma-7B (Azerbayev et al., 2023) and DeepSeek-7B (Shao et al., 2024) as the generators. Our results indicate that TSMC consistently improves both the quality of the generated solutions and the overall verification performance. Additionally, we empirically validate the theoretical advantage of TSMC as an unbiased estimator with reduced variance, further highlighting its effectiveness.

Our main contributions can be summarized as follows:

1. We propose a novel method based on TSMC that enhances the sampling efficiency of verification and reduces the reliance on process supervision obtained through human annotations or tree search in training verifiers.
2. We introduce a new theoretical framework for analyzing verification methods, providing deeper insights into their effectiveness and limitations.
3. Our empirical results demonstrate that TSMC consistently outperforms existing verification methods across multiple math benchmarks, utilizing various generators.

2 PRELIMINARIES

2.1 LLMs FOR MATH

Following Lightman et al. (2024), we fix the generator without further fine-tuning via reinforcement learning. For a problem statement \mathbf{x}_0 , a (tokenized) candidate solution can be sampled from the generator, denoted as $\mathbf{x}_{1:T} \sim p(\cdot | \mathbf{x}_0)$. For simplicity, we always assume the dependence on \mathbf{x}_0 and

no longer explicitly write it out in the following text. The solution is assumed to be decomposable as $\mathbf{x}_{1:T} = [\mathbf{x}_1, \dots, \mathbf{x}_T]$, where \mathbf{x}_i is a variable-length reasoning step. By default, the LLM generates all steps in an autoregressive manner, i.e., $\mathbf{x}_t \sim p(\cdot | \mathbf{x}_{1:t-1})$. Each solution $\mathbf{x}_{1:T}$ contains the reasoning process and an answer to the problem, with examples shown in Appendix F. We represent the extracted answer from the solution as $a = \text{Ans}(\mathbf{x}_{1:T})$, and its correctness as $\phi(a)$, which is 1 if it is correct (matched with the ground-truth answer) and 0 otherwise.

The primary methods for solving math problems with LLMs include majority voting (Wang et al., 2023e) and verification (Cobbe et al., 2021; Uesato et al., 2022; Lightman et al., 2024).

Majority Voting. Majority voting independently samples N (tokenized) candidate solutions $\{\mathbf{x}_{1:T}^i\}_{i=1}^N$ from the generator. It selects the final answer as the one with the most votes, i.e., $a^* = \arg \max_a \sum_{i=1}^N \mathbb{I}(a_i = a)$, where $\mathbb{I}(\cdot)$ is the indicator function.

Verification. Verification introduces an external verifier $r(\cdot)$ to evaluate the N solutions produced by the LLM generator. Existing methods can be roughly divided into two kinds: the outcome reward model (ORM) family and the process reward model (PRM) family. ORM directly evaluates the confidence score for each full solution as $s = r_{ORM}(\mathbf{x}_{1:T})$, while PRM instead aggregates the confidence scores of sub-sequences as $s = r_{PRM}(\mathbf{x}_{1:T}) = \text{Aggr}(\{r_{PRM}(\mathbf{x}_t | \mathbf{x}_{1:t-1})\}_{t=1}^T)$. Here, $r_{PRM}(\mathbf{x}_t | \mathbf{x}_{1:t-1})$ corresponds to the process reward, and $\text{Aggr}(\cdot)$ is the aggregation function such as the minimum or product:

$$\min = \min\{r_{PRM}(\mathbf{x}_t | \mathbf{x}_{1:t-1})_{t=1}^T\}, \quad \text{prod} = \prod_{t=1}^T r_{PRM}(\mathbf{x}_t | \mathbf{x}_{1:t-1}). \quad (1)$$

The final answer could either be selected from the solution with the highest score $a^* = \arg \max_{a^i} s^i$ (best-of- N), or the answer with the highest total weight $a^* = \arg \max_a \sum_{i=1}^N s^i \mathbb{I}(a_i = a)$ (weighted majority voting) (Li et al., 2023b). In this work, we mainly develop our method on top of the weighted majority voting due to its empirical better performance (Sun et al., 2024).

2.2 IMPORTANCE SAMPLING AND TWISTED SEQUENTIAL MONTE CARLO

Importance Sampling. Consider a target distribution $\sigma(\mathbf{x}_{1:T}) = \frac{\tilde{\sigma}(\mathbf{x}_{1:T})}{Z^\sigma}$, where $\tilde{\sigma}(\mathbf{x}_{1:T}) \geq 0$ is the unnormalized probability density and $Z^\sigma = \int_{\mathbf{x}_{1:T}} \tilde{\sigma}(\mathbf{x}_{1:T}) d\mathbf{x}_{1:T}$ is the normalizing factor, typically intractable. For a given function $h(\mathbf{x}_{1:T})$, it could be difficult to estimate its expectation under $\sigma(\mathbf{x}_{1:T})$ via directly sampling. Importance sampling (IS) (Robert & Casella, 2000) instead introduces a proposal distribution $q(\mathbf{x}_{1:T})$ and provides an estimator of the expectation as

$$\mathbb{E}_{\sigma(\mathbf{x}_{1:T})}[h(\mathbf{x}_{1:T})] = \frac{1}{Z^\sigma} \mathbb{E}_{q(\mathbf{x}_{1:T})}\left[\frac{\tilde{\sigma}(\mathbf{x}_{1:T})}{q(\mathbf{x}_{1:T})} h(\mathbf{x}_{1:T})\right] = \frac{\mathbb{E}_{q(\mathbf{x}_{1:T})}\left[\frac{\tilde{\sigma}(\mathbf{x}_{1:T})}{q(\mathbf{x}_{1:T})} h(\mathbf{x}_{1:T})\right]}{\mathbb{E}_{q(\mathbf{x}_{1:T})}\left[\frac{\tilde{\sigma}(\mathbf{x}_{1:T})}{q(\mathbf{x}_{1:T})}\right]}. \quad (2)$$

Here, $\frac{\tilde{\sigma}(\mathbf{x}_{1:T})}{q(\mathbf{x}_{1:T})}$ is known as the *importance weight* $w(\mathbf{x}_{1:T})$. Using some $q(\mathbf{x}_{1:T})$ that is easy to sample from, we can leverage Equation 2 to estimate the expectation via the Monte Carlo method,

$$\mathbb{E}_{\sigma(\mathbf{x}_{1:T})}[h(\mathbf{x}_{1:T})] \approx \sum_{i=1}^N \frac{w(\mathbf{x}_{1:T}^i)}{\sum_{j=1}^N w(\mathbf{x}_{1:T}^j)} h(\mathbf{x}_{1:T}^i), \quad \mathbf{x}_{1:T}^i \sim q(\mathbf{x}_{1:T}). \quad (3)$$

Although ideally a zero variance of the importance weight could be achieved when $q(\mathbf{x}_{1:T}) = \sigma(\mathbf{x}_{1:T})$, such a case rarely holds in practice. Remarkably, the distribution mismatches at each step are accumulated as the generation proceeds, leading to an exponentially increasing variance with respect to T (Doucet & Johansen, 2009). Such a limitation makes IS inefficient in the high-dimensional space since extensive sampling is needed to reduce the variance.

Twisted Sequential Monte Carlo. Twisted Sequential Monte Carlo (TSMC) enhances the sampling efficiency of IS by modifying the marginal distribution of the proposal, $q(\mathbf{x}_{1:t})$, to a more informative intermediate distribution, $\pi_t(\mathbf{x}_{1:t})$. The aim is to ensure that partial sequences from $\pi_t(\mathbf{x}_{1:t})$ are more likely to result in higher-quality samples in the final target distribution $\sigma(\mathbf{x}_{1:T})$. Here, $\{\pi_t\}_{t=1}^T$ is known as the (twisted) intermediate targets where $\pi_t(\mathbf{x}_{1:T}) = \frac{\tilde{\pi}_t(\mathbf{x}_{1:T})}{Z_t^\pi}$ and the

final target is aligned with $\tilde{\pi}_T \equiv \tilde{\sigma}$. In standard Sequential Monte Carlo, $\pi_t(\mathbf{x}_{1:t})$ is typically the marginal of the target distribution $\sigma(\mathbf{x}_{1:T})$, to ensure that at each time step, the marginal distribution matches the target. However, if our primary interest is only the final target $\sigma(\mathbf{x}_{1:T})$, we are free to design $\{\pi_t\}_{t=1}^{T-1}$ on the specific problem at hand, leading to the flexibility of the TSMC method.

TSMC operates recursively, alternating between generation and resampling. At each step, TSMC takes the input of N partial sequences, $\{\mathbf{x}_{1:t-1}^i\}_{i=1}^N$, following the distribution $\pi_{t-1}(\mathbf{x}_{1:t-1})$, and extends these sequences by sampling the next step from the proposal, i.e., $\mathbf{x}_t \sim q(\cdot|\mathbf{x}_{1:t-1})$. It computes the *incremental importance weight* for each sequence as

$$w_t(\mathbf{x}_{1:t}) = \frac{\tilde{\pi}_t(\mathbf{x}_{1:t})}{q(\mathbf{x}_t|\mathbf{x}_{1:t-1})\tilde{\pi}_{t-1}(\mathbf{x}_{1:t-1})}. \quad (4)$$

These weights are used to approximate the distribution π_t by resampling the partial sequences from a categorical distribution with the self-normalized weights:

$$\mathbf{x}_{1:t}^i \leftarrow \mathbf{x}_{1:t}^{\omega_i}, \quad \omega^i \sim \text{Cat}\left(\left\{\frac{w_t(\mathbf{x}_{1:t}^i)}{\sum_{j=1}^N w_t(\mathbf{x}_{1:t}^j)}\right\}_{i=1}^N\right), \quad i = 1, \dots, N. \quad (5)$$

This new set of N sequences would serve as the input to the next step of TSMC. With informative intermediate targets, the resampling step could promptly discard the sequences with a low potential in the target distribution and avoid a large variance in the importance weights. More importantly, since $\pi_T(\mathbf{x}_{1:T})$ is matched with the target $\sigma(\mathbf{x}_{1:T})$, TSMC always yields an unbiased estimator of $\sum_{\mathbf{x}_{1:T}} \tilde{\sigma}(\mathbf{x}_{1:T})h(\mathbf{x}_{1:T})$ regardless of the twist functions (Del Moral, 2004).

3 METHODOLOGY

3.1 EXISTING VERIFICATION METHODS ARE PERFORMING IMPORTANCE SAMPLING

The motivation of our method is based on the observation that existing verification methods are essentially performing IS. To see this, compare the normalized voting weight of each answer in majority voting and weighted majority voting when N is large

$$\lim_{N \rightarrow \infty} \frac{\sum_{i=1}^N \mathbb{I}(a_i = a)}{N} = \mathbb{E}_{p(\mathbf{x}_{1:T})}[\mathbb{I}(\text{Ans}(\mathbf{x}_{1:T}) = a)] \quad (\text{majority voting}) \quad (6)$$

$$\lim_{N \rightarrow \infty} \frac{\sum_{i=1}^N s_i \mathbb{I}(a_i = a)}{N} = \mathbb{E}_{p(\mathbf{x}_{1:T})}[r(\mathbf{x}_{1:T})\mathbb{I}(\text{Ans}(\mathbf{x}_{1:T}) = a)] \quad (\text{weighted majority voting}) \quad (7)$$

It can be seen that the weighting process actually introduces a factor $r(\mathbf{x}_{1:T})$ with a similar role of the importance weight in Equation 2. In particular, we can let $\tilde{\sigma}(\mathbf{x}_{1:T}) = p(\mathbf{x}_{1:T})r(\mathbf{x}_{1:T})$ and treat weighted majority voting as IS to estimate the answer voting weight

$$w(a) = \sum_{\mathbf{x}_{1:T}} \tilde{\sigma}(\mathbf{x}_{1:T})\mathbb{I}(\text{Ans}(\mathbf{x}_{1:T}) = a). \quad (8)$$

However, as described in Section 2.2, the importance weight in IS suffers from a large variance in the high-dimensional space, and so do the estimation objectives according to Proposition 3.1.

Proposition 3.1. *For IS with the target $\sigma(\mathbf{x}_{1:T})$ and proposal $q(\mathbf{x}_{1:T})$, up to a constant C independent of $q(\mathbf{x}_{1:T})$, the following identity in the variance holds for the set of all answers \mathcal{A} :*

$$\sum_{a \in \mathcal{A}} \mathbb{V}_q\left[\frac{\tilde{\sigma}(\mathbf{x}_{1:T})\mathbb{I}(\text{Ans}(\mathbf{x}_{1:T}) = a)}{q(\mathbf{x}_{1:T})}\right] = \mathbb{V}_q\left[\frac{\tilde{\sigma}(\mathbf{x}_{1:T})}{q(\mathbf{x}_{1:T})}\right] + C. \quad (9)$$

We include the proof in Appendix A.1. This issue also accounts for Problem I, i.e., plenty of samples are needed to reduce the variance of the estimator. Therefore, we aim to address this problem via TSMC, which provides the unbiased estimator of $w(a)$, but with less variance. We visualize the comparison between existing IS-based verification and our TSMC-based verification in Figure 1.

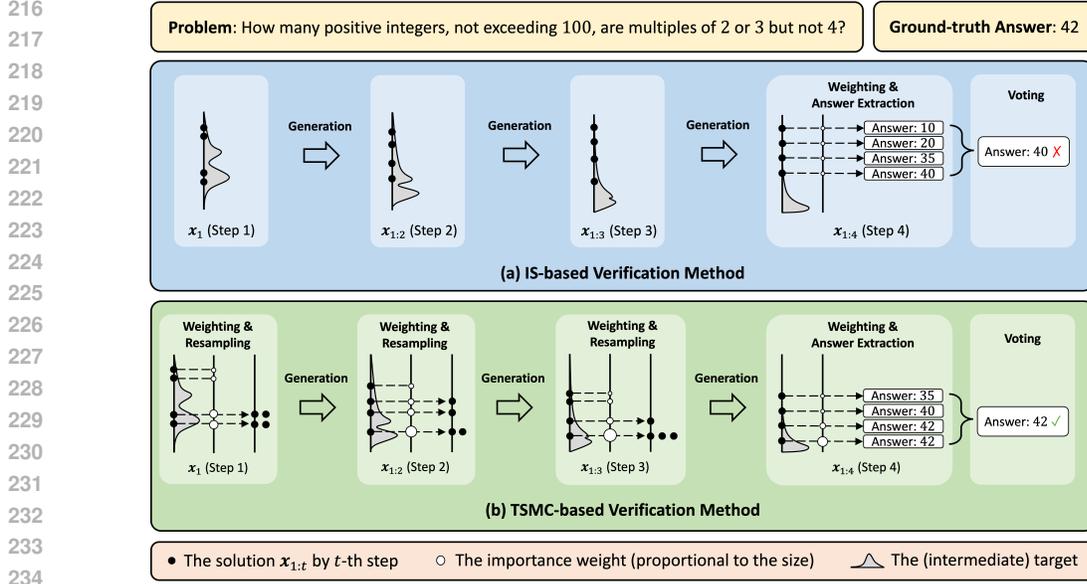


Figure 1: IS-based verification vs. TSMC-based verification: (a) **Typical IS-based verification** only weights (verifies) the solutions until they are fully generated, which often leads to generating incorrect solutions with high probability, aka low sampling efficiency. (b) **Our TSMC-based verification** weights and resamples partial solutions at each step of the generation process. This sequential resampling process reduces the discrepancy between the proposal and target distributions, improving the overall correctness of the generated solutions and thus the sampling efficiency.

3.2 VERIFICATION VIA TSMC

The optimal reward model in our problem is simply the correctness function of each solution, i.e., $r^*(\mathbf{x}_{1:T}) = \phi(\text{Ans}(\mathbf{x}_{1:T}))$. In the following section, we fix our target distribution as

$$\sigma(\mathbf{x}_{1:T}) = \frac{p(\mathbf{x}_{1:T})\phi(\text{Ans}(\mathbf{x}_{1:T}))}{Z_\sigma}, \quad (10)$$

since it corresponds to the actual target distribution we try to sample from.

To utilize TSMC for verification, we still need to decide the proposal distribution $q(\mathbf{x}_t|\mathbf{x}_{1:t-1})$ and intermediate targets $\{\pi_t\}_{t=1}^{T-1}$. Following Zhao et al. (2024), we define the intermediate targets through the twist functions $\{\psi_t\}_{t=1}^{T-1}$ where $\psi_t(\mathbf{x}_{1:t}) \geq 0$ are functions to be optimized

$$\pi_t(\mathbf{x}_{1:t}) = \frac{p(\mathbf{x}_{1:t})\psi_t(\mathbf{x}_{1:t})}{Z_t^\pi}. \quad (11)$$

Let $\psi_0(\mathbf{x}_0) \equiv 1$ and $\psi_T(\mathbf{x}_{1:T}) \equiv \phi(\text{Ans}(\mathbf{x}_{1:T}))$, the incremental importance weight is given by

$$w_t(\mathbf{x}_{1:t}) = \frac{p(\mathbf{x}_t|\mathbf{x}_{1:t-1})\psi_t(\mathbf{x}_{1:t})}{q(\mathbf{x}_t|\mathbf{x}_{1:t-1})\psi_{t-1}(\mathbf{x}_{1:t-1})}. \quad (12)$$

Zhao et al. (2024) have shown that the optimal proposal and intermediate targets correspond to

$$q_t^*(\mathbf{x}_t|\mathbf{x}_{1:t-1}) = \frac{\sigma(\mathbf{x}_{1:t})}{\sigma(\mathbf{x}_{1:t-1})} \quad \text{and} \quad \pi_t^*(\mathbf{x}_{1:t}) = \sigma(\mathbf{x}_{1:t}), \quad (13)$$

where $\sigma(\mathbf{x}_{1:t}) = \sum_{\mathbf{x}_{t+1:T}} \sigma(\mathbf{x}_{1:T})$ stands for the target marginal distribution. However, it is hard to directly apply these optimal choices to our case. We outline the reason and our approach as follows.

Proposal. There are two challenges in preventing us from using the optimal proposal. First, the combinatorial nature of the step \mathbf{x}_t , which consists of multiple tokens, makes $q_t^*(\mathbf{x}_t|\mathbf{x}_{1:t-1})$ generally intractable. Moreover, even if we can approximate it via finetuning, we still encounter the second challenge as the large variance in $q_t^*(\mathbf{x}_t|\mathbf{x}_{1:t-1})$ caused by the high dimensionality of \mathbf{x}_t .

This would result in the *weight degeneracy* issue (Naesseth et al., 2019) of TSMC, where the incremental importance weights would be dominated by a single sample, resulting in a poor diversity of solutions after resampling. We therefore simply let $q(\mathbf{x}_t|\mathbf{x}_{1:t-1}) = p(\mathbf{x}_t|\mathbf{x}_{1:t-1})$ for the ease of sampling and weight degeneracy would be alleviated when $q(\mathbf{x}_t|\mathbf{x}_{1:t-1})$ cancels in Equation 12.

Intermediate targets. The intermediate targets in Equation 13 is no longer optimal when we use $p(\mathbf{x}_t|\mathbf{x}_{1:t-1})$ as our proposal. However, it is also hard to solve the globally optimal intermediate targets for an arbitrary proposal. We instead seek to sequentially derive the locally optimal twists in a greedy manner. Since our ultimate goal is to estimate the answer weights $w(a)$, we start by looking for the optimal intermediate target $\pi_{T-1}^*(\mathbf{x}_{1:T-1})$ in minimizing the variance of the incremental importance weight in the last TSMC step. We prove the following proposition in Appendix A.2.

Proposition 3.2. *Given an (intermediate) target $\pi_t(\mathbf{x}_{1:t})$ and the proposal $q(\mathbf{x}_t|\mathbf{x}_{1:t-1})$, the optimal $\pi_{t-1}(\mathbf{x}_{1:t-1})$ in minimizing the variance of the incremental importance weight corresponds to*

$$\pi_{t-1}^q(\mathbf{x}_{1:t-1}) \propto \sqrt{\sum_{\mathbf{x}_t} \frac{\pi_t(\mathbf{x}_{1:t})^2}{q(\mathbf{x}_t|\mathbf{x}_{1:t-1})}}. \quad (14)$$

Taking $t = T$ and $q = p$ implies $\pi_{T-1}^p(\mathbf{x}_{1:T-1}) \propto p(\mathbf{x}_{1:T-1}) \sqrt{\sum_{\mathbf{x}_T} p(\mathbf{x}_T|\mathbf{x}_{1:T-1}) \phi(\text{Ans}(\mathbf{x}_{1:T}))}$. Here we apply the fact that $\phi(\text{Ans}(\mathbf{x}_{1:T}))^2 = \phi(\text{Ans}(\mathbf{x}_{1:T}))$ as it is binary. If we fix the intermediate target as the choice above, we could further propagate the derivation to previous steps by recursively applying Proposition 3.2, getting the locally optimal intermediate targets for $t < T$ as

$$\pi_t^p(\mathbf{x}_{1:t}) \propto p(\mathbf{x}_{1:t}) \sqrt{\sum_{\mathbf{x}_{t+1:T}} p(\mathbf{x}_{t+1:T}|\mathbf{x}_{1:t}) \phi(\text{Ans}(\mathbf{x}_{1:T}))}. \quad (15)$$

In particular, $\sum_{\mathbf{x}_{t+1:T}} p(\mathbf{x}_{t+1:T}|\mathbf{x}_{1:t}) \phi(\text{Ans}(\mathbf{x}_{1:T}))$ actually represents the value function $V^p(\mathbf{x}_{1:t})$ in reinforcement learning (Ouyang et al., 2022). Hence, the locally optimal twists are given by

$$\psi_t^p(\mathbf{x}_{1:t}) \propto \sqrt{V^p(\mathbf{x}_{1:t})}. \quad (16)$$

3.3 CONNECTION WITH THE PRM

Based on our above choices, the incremental importance weights in Equation 12 becomes

$$w_t^p(\mathbf{x}_{1:t}) = \frac{\psi_t^p(\mathbf{x}_{1:t})}{\psi_{t-1}^p(\mathbf{x}_{1:t-1})} \propto \sqrt{\frac{V^p(\mathbf{x}_{1:t})}{V^p(\mathbf{x}_{1:t-1})}}. \quad (17)$$

The incremental importance weight could also be treated as a measurement of the step quality, similar to the process reward in the PRM. To further augment this connection, note that

$$\prod_{t=1}^T w^p(\mathbf{x}_{1:t}) = \prod_{t=1}^T \frac{\psi_t^p(\mathbf{x}_{1:t})}{\psi_{t-1}^p(\mathbf{x}_{1:t-1})} = \frac{\psi_T^p(\mathbf{x}_{1:T})}{\psi_0^p(\mathbf{x}_0)} = \phi(\text{Ans}(\mathbf{x}_{1:T})), \quad (18)$$

which is in the same format as the PRM with `prod` aggregation. The key observation here is that TSMC always yields an unbiased estimator of the importance weight $\phi(\text{Ans}(\mathbf{x}_{1:T}))$ when there is no estimation error of V^p . We continue to compare this estimator to some existing PRMs.

The PRM learnt through automatic supervision. This class of PRMs (Wang et al., 2023b; Luo et al., 2024) computes the process reward by evaluating the value function at each partial solution with respect to a roll-out policy μ . The solution confidence score will be computed as

$$r_{PRM}(\mathbf{x}_{1:T}) = \text{Aggr}(\{r(\mathbf{x}_t|\mathbf{x}_{1:t-1})\}_{t=1}^T) = \text{Aggr}(\{V^\mu(\mathbf{x}_{1:t})\}_{t=1}^T). \quad (19)$$

But such an estimator is always biased no matter `min` or `prod` is used for aggregation.

The PRM learnt through human supervision. The human supervision is generated through the logical sense of the step correctness. We formally establish the definition of the step correctness in Definition A.1 and prove the following proposition in Appendix A.3.

Proposition 3.3. *The ground-truth PRM over the step correctness corresponds to*

$$r_{PRM}(\mathbf{x}_t|\mathbf{x}_{1:t-1}) = \mathbb{I}(\sigma(\mathbf{x}_{1:t}) > 0). \quad (20)$$

Therefore, the solution confidence score of this PRM is always an unbiased estimator of $\phi(\text{Ans}(\mathbf{x}_{1:T}))$ for both `min` and `prod` aggregation. However, using $\mathbb{I}(\sigma(\mathbf{x}_{1:t}) > 0)$ for the intermediate target, as tried by Uesato et al. (2022), does not effectively reduce the sampling variance or improve the verification performance since it ignores the likelihood of the proposal $p(\mathbf{x}_{t+1:T}|\mathbf{x}_{1:t})$.

3.4 VALUE FUNCTION ESTIMATION

The approximation of $\{\psi_t^p\}_{t=1}^{T-1}$ and r^* can be consolidated into a single learning task: estimating the value function V^p . We therefore use a single neural model parameterized by θ for approximation. Estimating the value function through independently sampled data from the policy (generator) is a well-studied topic (Bertsekas, 2012). It therefore eliminates the need for explicit process supervision during training, as outlined in Problem II.

In this paper, we adopt the Contrastive Twist Learning (CTL) method developed by Zhao et al. (2024). Directly approximating π_t^p would be hard, so we still approximate the target marginal $\sigma(\mathbf{x}_{1:t})$ to learn the value function and take the square root of the value function during the inference time. Let V^θ be our approximation of the value function V^p , and define the intermediate target $\pi_t^\theta(\mathbf{x}_{1:t}) = \frac{p(\mathbf{x}_{1:t})V^\theta(\mathbf{x}_{1:t})}{Z_t^\theta(\mathbf{x}_{1:t})}$. CTL minimizes the KL divergence between the target marginal distributions and the intermediate targets, i.e.,

$$\min_{\theta} L_{CTL}(\theta) = \min_{\theta} \sum_{t=1}^T D_{KL}(\sigma(\mathbf{x}_{1:t}) \parallel \pi_t^\theta(\mathbf{x}_{1:t})), \quad (21)$$

whose gradient at t -th step can be derived as

$$\mathbb{E}_{\sigma(\mathbf{x}_{1:t})}[\nabla_{\theta} \log V^\theta(\mathbf{x}_{1:t})] - \mathbb{E}_{\pi_t^\theta(\mathbf{x}_{1:t})}[\nabla_{\theta} \log V^\theta(\mathbf{x}_{1:t})]. \quad (22)$$

We approximate the gradient in the first term via rejection sampling while the gradient in the second term via importance sampling, as done in Equation 3. We include more training details in Appendix C.2 and summarize our entire TSMC-based verification algorithm in Appendix B.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

We outline the basic experimental setup in this section and include more details in Appendix C.

Datasets. Building on prior work (Uesato et al., 2022; Lightman et al., 2024; Wang et al., 2023b), we assess our TSMC method using two widely used math datasets: GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021). For GSM8K, we evaluate model performance across all testing instances. Whereas for MATH, we follow Lightman et al. (2024) to select a representative subset of 500 testing instances, referred to as ‘‘MATH500’’ in the following text.

Generators. We fine-tune our solution generators using two different pretrained LLMs, Llemma-7B (Azerbayev et al., 2023) and DeepSeek-7B (Shao et al., 2024). Following Sun et al. (2024), we use the filtered PRM800K (Lightman et al., 2024) as the supervised fine-tuning dataset.

Baselines. We compare our method to both the non-verification methods, including zero-shot greedy decoding and majority voting (MV) (Wang et al., 2023e), and the verification methods using weighted majority voting (WMV). For verification methods, we utilize various types of verifiers, including the ORM, the PRM trained with human supervision on PRM800K (Lightman et al., 2024)

and the PRM trained with automatic supervision on MATH-SHEPHERD (Wang et al., 2023b). We employ `min` for aggregation on both PRMs. We keep the architecture and pretrained weights of all the verifiers, including the reward model and the parameterized value function, the same as the generator across all settings. The problem solving rate (in %) is used as the metric for comparison.

TSMC details. Our TSMC is applied on the step level. We implement a warm-up stage that skips resampling in the initial stage, setting this threshold at 50 tokens across all experiments. A maximum of five resampling steps is allowed to reduce the latency. For sequences that terminate early, we assign an incremental importance weight of 1 during the remaining resampling steps. We employ stratified sampling (Kitagawa, 1996) for resampling to reduce the variance. Instead of resampling across the full batch of N solutions, we perform resampling over a mini-batch with M samples. The batch size M is fixed as 80 (the maximum number of sequences fit into our GPUs) by default.

4.2 MAIN RESULTS

To verify if TSMC actually improves the sampling efficiency with a better solution quality, we also directly perform the majority voting on the solutions generated by TSMC. We denote this method as TSMC + MV and the full TSMC as TSMC + WMV. We present our main results in Table 1.

Table 1: Comparative results in the problem solving rate (%) on GSM8K and MATH500 datasets. Llemma-7B and DeepSeek-8B are used as generators. We bold the best results in each category. The voting is performed on 240 samples.

Generators	Methods	GSM8K	MATH500
Llemma-7B	Greedy	38.2	19.4
	MV	72.5	41.2
	WMV w. ORM	78.7	43.0
	WMV w. PRM (PRM800K)	73.6	43.2
	WMV w. PRM (SHEPHERD)	79.2	43.6
	TSMC + MV (Ours)	77.6	44.0
	TSMC + WMV (Ours)	80.6	45.6
DeepSeek-7B	Greedy	61.2	30.8
	MV	86.4	52.8
	WMV w. ORM	86.6	55.0
	WMV w. PRM (PRM800K)	87.0	55.2
	WMV w. PRM (SHEPHERD)	89.5	52.6
	TSMC + MV (Ours)	88.7	54.8
	TSMC + WMV (Ours)	90.8	56.2

It is evident that TSMC + MV demonstrates a significant improvement over vanilla MV, highlighting its effectiveness in enhancing the overall solution quality. This advantage is particularly pronounced when using the relatively weaker generator Llemma-7B, which corresponds to cases where the proposal distribution deviates significantly from the target. Moreover, TSMC consistently outperforms other methods in terms of final verification performance. It is worth noting that the final verification step in TSMC operates independently of the generator, meaning that a better reward model could further improve TSMC’s performance in WMV. Overall, our TSMC-based verification method shows a clear advantage over existing verification methods, with a simpler training target.

4.3 IMPACT OF THE BIAS IN THE ESTIMATOR

TSMC is characterized by its unbiased estimation of the importance weight, which is $\phi(\text{Ans}(\mathbf{x}_{1:T}))$ in our task. However, since the training error is unavoidable in practice, it remains unclear whether such kind of unbiased estimation at the theoretical optimal case is useful. We thus look into this problem by comparing different biased and unbiased estimators analyzed in Section 3.3.

We consider both the PRM predicting the step correctness, PRM (PRM800K), and the PRM predicting the value function, including PRM (SHEPHERD) and the value function estimated in TSMC. Beyond the `min` and `prod` aggregations, we also consider an unbiased ORM-like strategy that only

uses the process reward at the last step as the solution score:

$$\text{last} = r_{PRM}(\mathbf{x}_T | \mathbf{x}_{1:T-1}). \tag{23}$$

Exceptionally, we evaluate the value function of TSMC on the data generated by TSMC, so we use the last incremental importance weight for `last` in this scenario as we do in our original TSMC method. This is equivalent to the product of incremental importance weights, which is also an unbiased estimator. We compare all estimators in Figure 2.

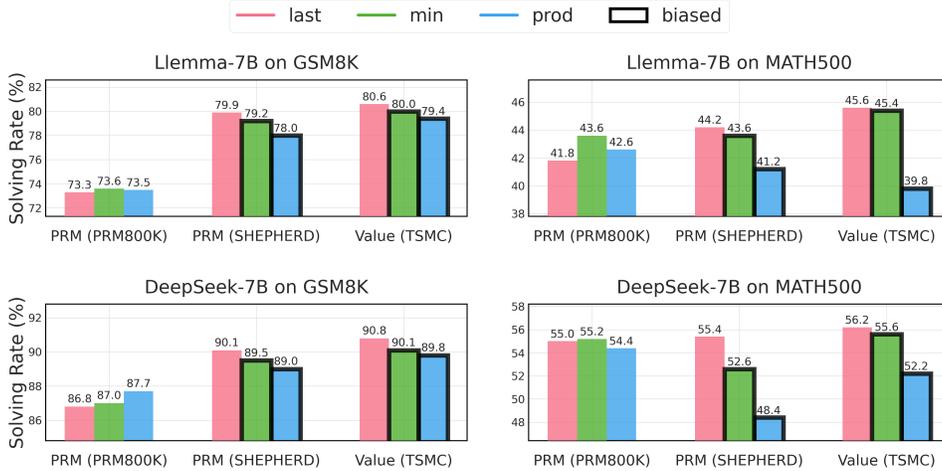


Figure 2: Comparison among all biased and unbiased estimators of the importance weight.

The trend is highly consistent when the value function is used as the process reward, as in PRM (SHEPHERD) and Value (TSMC). The `prod` strategy shows an overall inferior performance across all settings. Since the step value could be small, the product of step values is highly biased towards the solutions with fewer steps. In contrast, `min` could overcome such a bias as its value is insensitive to the number of steps. It also shows a clear advantage in comparison to `prod`, in line with the choice of (Wang et al., 2023b). However, its performance is still consistently worse than the unbiased estimator using `last`. A different pattern shows up in PRM (PRM800K), where the step correctness is used as the process reward. The `min` strategy still achieves the overall best result, but `prod` is also comparably good. We find no advantage of `last` in this case as all three estimators are unbiased. Instead, `prod` and `min` would benefit from less modeling error with its ensemble classifier.

We find our results consistent with the observation from Sun et al. (2024). Basically, the advantage of PRM against ORM holds only when both of them are unbiased estimators of $\phi(\text{Ans}(\mathbf{x}_{1:T}))$. When the PRM is biased, there is no clear guarantee of a better performance against the ORM, which is always an unbiased estimator. While TSMC assimilates the strengths of the unbiased estimation from the ORM and the intermediate step modeling from the PRM, leading to the best performance.

4.4 IMPACT OF THE VARIANCE IN THE ESTIMATOR

Besides of being unbiased, TSMC reduces the variance of the importance weight via informative twist functions. To investigate the impact of the variance, we consider the following TSMC variants: using the step correctness predicted by PRM (PRM800K) as the incremental importance weight; using the process reward in PRM (SHEPHERD) for the estimation of the value function; and using $V^p(\mathbf{x}_{1:t})$ rather than $\sqrt{V^p(\mathbf{x}_{1:t})}$ for the twist, which approximates the target marginal $\sigma(\mathbf{x}_{1:t})$ as the intermediate target. Using Llemma-7B on MATH500 as the example, we examine the performance of these variants in Figure 3.

As observed by Uesato et al. (2022), using the step correctness in the intermediate decoding does not bring any improvement since the likelihood from the generator is ignored. The value function provided by PRM (SHEPHERD) is more informative and leads to a better performance. But since the value function is evaluated on a different generator μ , it still achieves a worse performance than our TSMC using the approximated V^p as the twist. Finally, using $V^p(\mathbf{x}_{1:t})$ as the twist also leads to an inferior performance than using $\sqrt{V^p(\mathbf{x}_{1:t})}$, indicating the necessity to optimize the variance.

486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539

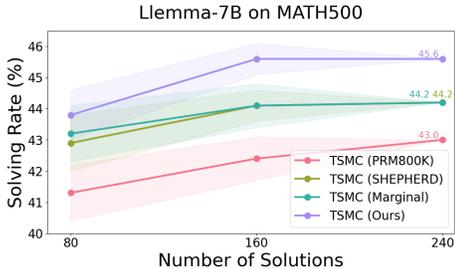


Figure 3: TSMC with different intermediate targets. Variance are visualized across many subsamples of the 240 solutions per problem.

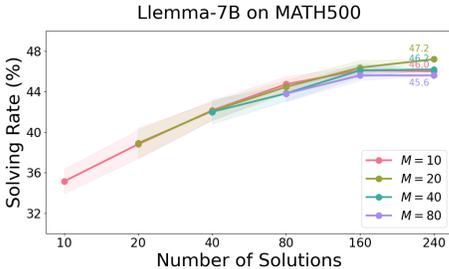


Figure 4: Ablation study on the TSMC batch size. Variance are visualized across many subsamples of the 240 solutions per problem.

4.5 SENSITIVITY ANALYSIS OF THE TSMC BATCH SIZE

In above experiments, we use the maximum number of samples that can fit into the memory of our GPUs as the batch size M of TSMC. In practice, a smaller memory and or a larger model would lead to a smaller TSMC batch size. We thus try to investigate the effect of M on the TSMC performance by varying M over the values in $\{10, 20, 40, 80\}$. The comparative results are shown in Figure 4. It can be seen that smaller M achieves comparable and even better results. The advantage of TSMC still clearly holds when $M = 10$, which is a batch size that can be fit into most GPUs.

5 RELATED WORK

Verification for reasoning. Verification has proven to be an effective approach for enhancing the multi-step reasoning ability of LLMs. Two widely adopted verification methods are the Outcome Reward Model (ORM) (Cobbe et al., 2021) and the Process Reward Model (PRM) (Uesato et al., 2022). While empirical evidence suggests that PRM outperforms ORM (Lightman et al., 2024), training the PRM presents a significant challenge due to the need for process supervision, which is often difficult to obtain. Recent research has therefore increasingly focused on automatic supervision to train PRMs more efficiently (Wang et al., 2023d; Luo et al., 2024; Wang et al., 2024).

(Twisted) Sequential Monte Carlo. Sequential Monte Carlo (SMC) is a generic statistical inference approach that has been widely applied across various domains, including signal processing (Doucet & Johansen, 2009; Simon J Godsill & West, 2004), financial econometrics (Johannes & Polson, 2010; Creal, 2012), and robotics (Montemerlo et al., 2002; Bailey & Durrant-Whyte, 2006; Thrun et al., 2005). Recently, SMC has been integrated with neural models to enhance sequential generative models, such as diffusion models (Trippe et al., 2023; Wu et al., 2023) and LLMs (Lew et al., 2023; Zhao et al., 2024). The most relevant work to ours is Zhao et al. (2024), which presents a general framework for controlled text generation using Twisted Sequential Monte Carlo (TSMC). Our work primarily focuses on multi-step reasoning, one of the most challenging areas for LLMs. Additionally, we are the first to bridge TSMC with the predominant verification methods, offering a novel theoretical perspective for explainability.

6 CONCLUSION & LIMITATION

In this paper, we introduce a novel verification method for multi-step reasoning using Twisted Sequential Monte Carlo (TSMC). Our approach sequentially approximates intermediate targets, enhancing the reasoning process of large language models and improving both solution quality and sampling efficiency. By incorporating step-wise guidance without human supervision for training, our method provides a scalable framework for various multi-step reasoning tasks.

Although promising, our method also introduces the additional latency in the inference time due to variable step lengths. A potential optimization involves blockwise resampling over a fixed number of tokens. Future work could also explore the impact of TSMC batch size and refine algorithmic design for further efficiency gains.

REFERENCES

- 540
541
542 Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen McAleer, Al-
543 bert Q Jiang, Jia Deng, Stella Biderman, and Sean Welleck. Llemma: An open language model
544 for mathematics. *arXiv preprint arXiv:2310.10631*, 2023.
- 545 T. Bailey and H. Durrant-Whyte. Simultaneous localization and mapping (slam): part ii. *IEEE*
546 *Robotics & Automation Magazine*, 13(3):108–117, 2006. doi: 10.1109/MRA.2006.1678144.
547
- 548 D. Bertsekas. *Dynamic Programming and Optimal Control: Volume II; Approximate Dynamic Pro-*
549 *gramming*. Athena Scientific optimization and computation series. Athena Scientific, 2012. ISBN
550 9781886529441. URL <https://books.google.com/books?id=C1JEEAAAQBAJ>.
- 551 Mark Briers, Arnaud Doucet, and Simon Maskell. Smoothing algorithms for state–space mod-
552 els. *Annals of the Institute of Statistical Mathematics*, 62:61–89, 02 2009. doi: 10.1007/
553 s10463-009-0236-2.
554
- 555 Guoxin Chen, Minpeng Liao, Chengxi Li, and Kai Fan. Alphamath almost zero: process supervision
556 without process, 2024. URL <https://arxiv.org/abs/2405.03553>.
- 557 Liang Chen, Yichi Zhang, Shuhuai Ren, Haozhe Zhao, Zefan Cai, Yuchi Wang, Peiyi Wang, Tianyu
558 Liu, and Baobao Chang. Towards end-to-end embodied decision making via multi-modal large
559 language model: Explorations with gpt4-vision and beyond. *arXiv preprint arXiv:2310.02071*,
560 2023.
561
- 562 N. Chopin and O. Papaspiliopoulos. *An Introduction to Sequential Monte Carlo*. Springer Series
563 in Statistics. Springer International Publishing, 2020. ISBN 9783030478452. URL <https://books.google.com/books?id=ZZEAEAAAQBAJ>.
- 564
565 Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser,
566 Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to
567 solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.
568
- 569 Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *International*
570 *conference on computers and games*, pp. 72–83. Springer, 2006.
571
- 572 Drew Creal. A survey of sequential monte carlo methods for economics and finance. *Econometric*
573 *Reviews*, 31(3):245–296, 2012. doi: 10.1080/07474938.2011.607333. URL <https://doi.org/10.1080/07474938.2011.607333>.
574
- 575 Pierre Del Moral. *Feynman-Kac Formulae: Genealogical and Interacting Particle Systems With*
576 *Applications*, volume 100. 05 2004. ISBN 0387202684. doi: 10.1007/978-1-4684-9393-1.
577
- 578 Pierre Del Moral, Arnaud Doucet, and Ajay Jasra. Sequential Monte Carlo Samplers. *Journal of*
579 *the Royal Statistical Society Series B: Statistical Methodology*, 68(3):411–436, 05 2006. ISSN
580 1369-7412. doi: 10.1111/j.1467-9868.2006.00553.x. URL [https://doi.org/10.1111/
581 j.1467-9868.2006.00553.x](https://doi.org/10.1111/j.1467-9868.2006.00553.x).
- 582 A. Doucet, A. Smith, N. de Freitas, and N. Gordon. *Sequential Monte Carlo Methods in Practice*.
583 Information Science and Statistics. Springer New York, 2001. ISBN 9780387951461. URL
584 <https://books.google.com/books?id=uxX-koqKtMMC>.
585
- 586 Arnaud Doucet and Adam Johansen. A tutorial on particle filtering and smoothing: Fifteen years
587 later. *Handbook of Nonlinear Filtering*, 12, 01 2009.
- 588 Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song,
589 and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv*
590 *preprint arXiv:2103.03874*, 2021.
591
- 592 Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song,
593 and Denny Zhou. Large language models cannot self-correct reasoning yet. *arXiv preprint*
arXiv:2310.01798, 2023.

- 594 Michael Johannes and Nicholas Polson. Chapter 13 - mcmc methods for continuous-time financial
595 econometrics. In YACINE AÏT-SAHALIA and LARS PETER HANSEN (eds.), *Handbook of*
596 *Financial Econometrics: Applications*, volume 2 of *Handbooks in Finance*, pp. 1–72. Elsevier,
597 San Diego, 2010. doi: <https://doi.org/10.1016/B978-0-444-53548-1.50003-9>. URL <https://www.sciencedirect.com/science/article/pii/B9780444535481500039>.
599
- 600 Jean Kaddour, Joshua Harris, Maximilian Mozes, Herbie Bradley, Roberta Raileanu, and
601 Robert McHardy. Challenges and applications of large language models. *arXiv preprint*
602 *arXiv:2307.10169*, 2023.
- 603 Genshiro Kitagawa. Monte carlo filter and smoother for non-gaussian nonlinear state space
604 models. *Journal of Computational and Graphical Statistics*, 5(1):1–25, 1996. doi: 10.
605 1080/10618600.1996.10474692. URL [https://www.tandfonline.com/doi/abs/](https://www.tandfonline.com/doi/abs/10.1080/10618600.1996.10474692)
606 [10.1080/10618600.1996.10474692](https://www.tandfonline.com/doi/abs/10.1080/10618600.1996.10474692).
- 607 Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference*
608 *on machine learning*, pp. 282–293. Springer, 2006.
- 609 Alexander K. Lew, Tan Zhi-Xuan, Gabriel Grand, and Vikash K. Mansinghka. Sequential monte
610 carlo steering of large language models using probabilistic programs, 2023. URL [https://](https://arxiv.org/abs/2306.03081)
611 arxiv.org/abs/2306.03081.
- 612 Lei Li, Yuwei Yin, Shicheng Li, Liang Chen, Peiyi Wang, Shuhuai Ren, Mukai Li, Yazheng Yang,
613 Jingjing Xu, Xu Sun, et al. M3it: A large-scale dataset towards multi-modal multilingual instruc-
614 tion tuning. *arXiv preprint arXiv:2306.04387*, 2023a.
- 615 Yifei Li, Zeqi Lin, Shizhuo Zhang, Qiang Fu, Bei Chen, Jian-Guang Lou, and Weizhu Chen. Mak-
616 ing language models better reasoners with step-aware verifier. In Anna Rogers, Jordan Boyd-
617 Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association*
618 *for Computational Linguistics (Volume 1: Long Papers)*, pp. 5315–5333, Toronto, Canada, July
619 2023b. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.291. URL
620 <https://aclanthology.org/2023.acl-long.291>.
621 <https://aclanthology.org/2023.acl-long.291>.
- 622 Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan
623 Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let’s verify step by step. In *The Twelfth*
624 *International Conference on Learning Representations*, 2024. URL [https://openreview.](https://openreview.net/forum?id=v8L0pN6EOi)
625 [net/forum?id=v8L0pN6EOi](https://openreview.net/forum?id=v8L0pN6EOi).
- 626 Liangchen Luo, Yinxiao Liu, Rosanne Liu, Samrat Phatale, Harsh Lara, Yunxuan Li, Lei Shu, Yun
627 Zhu, Lei Meng, Jiao Sun, and Abhinav Rastogi. Improve mathematical reasoning in language
628 models by automated process supervision, 2024. URL [https://arxiv.org/abs/2406.](https://arxiv.org/abs/2406.06592)
629 [06592](https://arxiv.org/abs/2406.06592).
- 630 Swaroop Mishra, Arindam Mitra, Neeraj Varshney, Bhavdeep Sachdeva, Peter Clark, Chitta Baral,
631 and Ashwin Kalyan. NumGLUE: A suite of fundamental yet challenging mathematical reason-
632 ing tasks. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Proceedings*
633 *of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long*
634 *Papers)*, pp. 3505–3523, Dublin, Ireland, May 2022. Association for Computational Linguis-
635 tics. doi: 10.18653/v1/2022.acl-long.246. URL [https://aclanthology.org/2022.](https://aclanthology.org/2022.acl-long.246)
636 [acl-long.246](https://aclanthology.org/2022.acl-long.246).
- 637 Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. Fastslam: a factored
638 solution to the simultaneous localization and mapping problem. In *Eighteenth National Con-*
639 *ference on Artificial Intelligence*, pp. 593–598, USA, 2002. American Association for Artificial
640 Intelligence. ISBN 0262511290.
- 641 Sidharth Mudgal, Jong Lee, Harish Ganapathy, YaGuang Li, Tao Wang, Yanping Huang, Zhifeng
642 Chen, Heng-Tze Cheng, Michael Collins, Trevor Strohman, Jilin Chen, Alex Beutel, and Ahmad
643 Beirami. Controlled decoding from language models. In *Forty-first International Conference on*
644 *Machine Learning*, 2024. URL <https://openreview.net/forum?id=bVIcZb7Qa0>.
- 645 Christian Naesseth, Fredrik Lindsten, and Thomas Schön. *Elements of Sequential Monte Carlo*. 01
646 2019. ISBN 978-1-68083-632-5. doi: 10.1561/9781680836332.

- 648 Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong
649 Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kel-
650 ton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike,
651 and Ryan Lowe. Training language models to follow instructions with human feedback, 2022.
652 URL <https://arxiv.org/abs/2203.02155>.
- 653 Joon Sung Park, Joseph O’Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and
654 Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings
655 of the 36th Annual ACM Symposium on User Interface Software and Technology*, pp. 1–22, 2023.
656
- 657 Christian Robert and George Casella. Monte carlo statistical method. *Technometrics*, 42, 11 2000.
658 doi: 10.2307/1270959.
- 659 Subhro Roy and Dan Roth. Solving general arithmetic word problems. In Lluís Màrquez,
660 Chris Callison-Burch, and Jian Su (eds.), *Proceedings of the 2015 Conference on Empiri-
661 cal Methods in Natural Language Processing*, pp. 1743–1752, Lisbon, Portugal, September
662 2015. Association for Computational Linguistics. doi: 10.18653/v1/D15-1202. URL <https://aclanthology.org/D15-1202>.
- 663
664
665 Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi
666 Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Ev-
667 timov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong,
668 Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier,
669 Thomas Scialom, and Gabriel Synnaeve. Code llama: Open foundation models for code, 2024.
670 URL <https://arxiv.org/abs/2308.12950>.
- 671
672 Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, Y.K. Li, and
673 Daya Guo Y. Wu. Deepseekmath: Pushing the limits of mathematical reasoning in open language
674 models, 2024. URL <https://arxiv.org/abs/2402.03300>.
- 675 Jianhao Shen, Yichun Yin, Lin Li, Lifeng Shang, Xin Jiang, Ming Zhang, and Qun Liu. Generate
676 & rank: A multi-task framework for math word problems. In Marie-Francine Moens, Xuanjing
677 Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Findings of the Association for Computa-
678 tional Linguistics: EMNLP 2021*, pp. 2269–2279, Punta Cana, Dominican Republic, November
679 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.findings-emnlp.195.
680 URL <https://aclanthology.org/2021.findings-emnlp.195>.
- 681
682 David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche,
683 Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering
684 the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.
- 685 Arnaud Doucet Simon J Godsill and Mike West. Monte carlo smoothing for nonlinear time se-
686 ries. *Journal of the American Statistical Association*, 99(465):156–168, 2004. doi: 10.1198/
687 016214504000000151. URL <https://doi.org/10.1198/016214504000000151>.
- 688
689 Yifan Song, Weimin Xiong, Dawei Zhu, Cheng Li, Ke Wang, Ye Tian, and Sujian Li. Restgpt: Con-
690 necting large language models with real-world applications via restful apis. corr, abs/2306.06624,
691 2023. doi: 10.48550. *arXiv preprint arXiv.2306.06624*.
- 692
693 Zhiqing Sun, Longhui Yu, Yikang Shen, Weiyang Liu, Yiming Yang, Sean Welleck, and Chuang
694 Gan. Easy-to-hard generalization: Scalable alignment beyond human supervision. *arXiv preprint
695 arXiv:2403.09472*, 2024.
- 696 Maciej Świechowski, Konrad Godlewski, Bartosz Sawicki, and Jacek Mańdziuk. Monte carlo tree
697 search: A review of recent modifications and applications. *Artificial Intelligence Review*, 56(3):
698 2497–2562, 2023.
- 699
700 S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. Intelligent Robotics and Autonomous
701 Agents series. MIT Press, 2005. ISBN 9780262201629. URL <https://books.google.com/books?id=2Zn6AQAAQBAJ>.

- 702 Brian L. Trippe, Jason Yim, Doug Tischer, David Baker, Tamara Broderick, Regina Barzilay, and
703 Tommi S. Jaakkola. Diffusion probabilistic modeling of protein backbones in 3d for the motif-
704 scaffolding problem. In *The Eleventh International Conference on Learning Representations*,
705 2023. URL <https://openreview.net/forum?id=6TxBxqNME1Y>.
706
- 707 Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia
708 Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process-and
709 outcome-based feedback. *arXiv preprint arXiv:2211.14275*, 2022.
- 710 Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan,
711 and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models.
712 *arXiv preprint arXiv:2305.16291*, 2023a.
- 713 Peiyi Wang, Lei Li, Liang Chen, Feifan Song, Binghuai Lin, Yunbo Cao, Tianyu Liu, and Zhi-
714 fang Sui. Making large language models better reasoners with alignment. *arXiv preprint*
715 *arXiv:2309.02144*, 2023b.
- 716 Peiyi Wang, Lei Li, Liang Chen, Dawei Zhu, Binghuai Lin, Yunbo Cao, Qi Liu, Tianyu Liu, and
717 Zhifang Sui. Large language models are not fair evaluators. *arXiv preprint arXiv:2305.17926*,
718 2023c.
- 719 Peiyi Wang, Lei Li, Zhihong Shao, RX Xu, Damai Dai, Yifei Li, Deli Chen, Y Wu, and Zhifang
720 Sui. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. *CoRR*,
721 *abs/2312.08935*, 2023d.
- 722 Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha
723 Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language
724 models. In *The Eleventh International Conference on Learning Representations*, 2023e. URL
725 <https://openreview.net/forum?id=1PL1NIMMrw>.
- 726 Zihan Wang, Yunxuan Li, Yuexin Wu, Liangchen Luo, Le Hou, Hongkun Yu, and Jingbo Shang.
727 Multi-step problem solving through a verifier: An empirical analysis on model-induced process
728 supervision, 2024. URL <https://arxiv.org/abs/2402.02658>.
- 729 Zeqiu Wu, Yushi Hu, Weijia Shi, Nouha Dziri, Alane Suhr, Prithviraj Ammanabrolu, Noah A Smith,
730 Mari Ostendorf, and Hannaneh Hajishirzi. Fine-grained human feedback gives better rewards for
731 language model training. *arXiv preprint arXiv:2306.01693*, 2023.
- 732 Stephen Zhao, Rob Brekelmans, Alireza Makhzani, and Roger Baker Grosse. Probabilistic infer-
733 ence in language models via twisted sequential Monte Carlo. In Ruslan Salakhutdinov, Zico
734 Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp
735 (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of
736 *Proceedings of Machine Learning Research*, pp. 60704–60748. PMLR, 21–27 Jul 2024. URL
737 <https://proceedings.mlr.press/v235/zhao24c.html>.
- 738 Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang,
739 Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and
740 chatbot arena. *arXiv preprint arXiv:2306.05685*, 2023.

747 A PROOFS

748 A.1 PROOF FOR ANSWER WEIGHT VARIANCE

749
750 **Proposition 3.1.** *For IS with the target $\sigma(\mathbf{x}_{1:T})$ and proposal $q(\mathbf{x}_{1:T})$, up to a constant C independ-*
751 *ent of $q(\mathbf{x}_{1:T})$, the following identity in the variance holds for the set of all answers \mathcal{A} :*

$$752 \sum_{a \in \mathcal{A}} \mathbb{V}_q \left[\frac{\tilde{\sigma}(\mathbf{x}_{1:T}) \mathbb{I}(\text{Ans}(\mathbf{x}_{1:T}) = a)}{q(\mathbf{x}_{1:T})} \right] = \mathbb{V}_q \left[\frac{\tilde{\sigma}(\mathbf{x}_{1:T})}{q(\mathbf{x}_{1:T})} \right] + C. \quad (9)$$

Proof. For simplicity, denote $f_a(\mathbf{x}_{1:T}) = \mathbb{I}(\text{Ans}(\mathbf{x}_{1:T}) = a)$. Using the fact that $f_a(\mathbf{x}_{1:T})^2 = f_a(\mathbf{x}_{1:T})$ and $\sum_{a \in \mathcal{A}} f_a(\mathbf{x}_{1:T}) = 1$, we have

$$\begin{aligned}
\sum_{a \in \mathcal{A}} \mathbb{V}_q \left[\frac{\tilde{\sigma}(\mathbf{x}_{1:T}) f_a(\mathbf{x}_{1:T})}{q(\mathbf{x}_{1:T})} \right] &= \sum_{a \in \mathcal{A}} \left(\mathbb{E}_q \left[\left(\frac{\tilde{\sigma}(\mathbf{x}_{1:T}) f_a(\mathbf{x}_{1:T})}{q(\mathbf{x}_{1:T})} \right)^2 \right] - \mathbb{E}_q \left[\frac{\tilde{\sigma}(\mathbf{x}_{1:T}) f_a(\mathbf{x}_{1:T})}{q(\mathbf{x}_{1:T})} \right]^2 \right) \\
&= \sum_{a \in \mathcal{A}} \left(\sum_{\mathbf{x}_{1:T}} \frac{\tilde{\sigma}(\mathbf{x}_{1:T})^2 f_a(\mathbf{x}_{1:T})^2}{q(\mathbf{x}_{1:T})} - \left(\sum_{\mathbf{x}_{1:T}} \tilde{\sigma}(\mathbf{x}_{1:T}) f_a(\mathbf{x}_{1:T}) \right)^2 \right) \\
&= \sum_{\mathbf{x}_{1:T}} \frac{\tilde{\sigma}(\mathbf{x}_{1:T})^2 \sum_a f_a(\mathbf{x}_{1:T})^2}{q(\mathbf{x}_{1:T})} - \sum_{a \in \mathcal{A}} \left(\sum_{\mathbf{x}_{1:T}} \tilde{\sigma}(\mathbf{x}_{1:T}) f_a(\mathbf{x}_{1:T}) \right)^2 \quad (24) \\
&= \sum_{\mathbf{x}_{1:T}} \frac{\tilde{\sigma}(\mathbf{x}_{1:T})^2}{q(\mathbf{x}_{1:T})} - \sum_{a \in \mathcal{A}} \left(\sum_{\mathbf{x}_{1:T}} \tilde{\sigma}(\mathbf{x}_{1:T}) f_a(\mathbf{x}_{1:T}) \right)^2 \\
&= \mathbb{V}_q \left[\frac{\tilde{\sigma}(\mathbf{x}_{1:T})}{q(\mathbf{x}_{1:T})} \right] + C.
\end{aligned}$$

Here, $C = \left(\sum_{\mathbf{x}_{1:T}} \tilde{\sigma}(\mathbf{x}_{1:T}) \right)^2 - \sum_{a \in \mathcal{A}} \left(\sum_{\mathbf{x}_{1:T}} \tilde{\sigma}(\mathbf{x}_{1:T}) f_a(\mathbf{x}_{1:T}) \right)^2$ is independent of $q(\mathbf{x}_{1:T})$. \square

A.2 PROOF FOR THE LOCALLY OPTIMAL INTERMEDIATE TARGET

Proposition 3.2. *Given an (intermediate) target $\pi_t(\mathbf{x}_{1:t})$ and the proposal $q(\mathbf{x}_t | \mathbf{x}_{1:t-1})$, the optimal $\pi_{t-1}(\mathbf{x}_{1:t-1})$ in minimizing the variance of the incremental importance weight corresponds to*

$$\pi_{t-1}^q(\mathbf{x}_{1:t-1}) \propto \sqrt{\sum_{\mathbf{x}_t} \frac{\pi_t(\mathbf{x}_{1:t})^2}{q(\mathbf{x}_t | \mathbf{x}_{1:t-1})}}. \quad (14)$$

Proof. Note that for the expectation of the importance weight, $\mathbb{E}_{p^{\pi_{t-1}}} \left[\frac{\pi_t(\mathbf{x}_{1:t})}{p(\mathbf{x}_t | \mathbf{x}_{1:t-1}) \pi_{t-1}(\mathbf{x}_{1:t-1})} \right] = 1$. So we have the variance of the importance weight as

$$\mathbb{V}_{p^{\pi_{t-1}}} \left[\frac{\pi_t(\mathbf{x}_{1:t})}{q(\mathbf{x}_t | \mathbf{x}_{1:t-1}) \pi_{t-1}(\mathbf{x}_{1:t-1})} \right] = \mathbb{E}_{q^{\pi_{t-1}}} \left[\left(\frac{\pi_t(\mathbf{x}_{1:t})}{q(\mathbf{x}_t | \mathbf{x}_{1:t-1}) \pi_{t-1}(\mathbf{x}_{1:t-1})} \right)^2 \right] - 1 \quad (25)$$

Minimizing the variance is thus equivalent to minimizing $\mathbb{E}_{q^{\pi_{t-1}}} \left[\left(\frac{\pi_t(\mathbf{x}_{1:t})}{p(\mathbf{x}_t | \mathbf{x}_{1:t-1}) \pi_{t-1}(\mathbf{x}_{1:t-1})} \right)^2 \right]$. Subject to the constraint of the probability, we introduce the Lagrange multiplier λ in our objective

$$\begin{aligned}
&\min_{\pi_{t-1}} \mathbb{E}_{q^{\pi_{t-1}}} \left[\left(\frac{\pi_t(\mathbf{x}_{1:t})}{q(\mathbf{x}_t | \mathbf{x}_{1:t-1}) \pi_{t-1}(\mathbf{x}_{1:t-1})} \right)^2 \right] + \lambda \left(\sum_{\mathbf{x}_{1:t-1}} \pi_{t-1}(\mathbf{x}_{1:t-1}) - 1 \right) \\
&= \min_{\pi_{t-1}} \sum_{\mathbf{x}_{1:t}} \frac{\pi_t(\mathbf{x}_{1:t})^2}{q(\mathbf{x}_t | \mathbf{x}_{1:t-1}) \pi_{t-1}(\mathbf{x}_{1:t-1})} + \lambda \left(\sum_{\mathbf{x}_{1:t-1}} \pi_{t-1}(\mathbf{x}_{1:t-1}) - 1 \right). \quad (26)
\end{aligned}$$

Taking $\frac{(\cdot)}{\pi_{t-1}(\mathbf{x}_{1:t-1})} = 0$, we get

$$-\sum_{\mathbf{x}_t} \frac{\pi_t(\mathbf{x}_{1:t})^2}{q(\mathbf{x}_t | \mathbf{x}_{1:t-1}) \pi_{t-1}^2(\mathbf{x}_{1:t-1})} + \lambda = 0. \quad (27)$$

This yields the optimal π_{t-1}^q given q as

$$\pi_{t-1}^q(\mathbf{x}_{1:t-1}) = \frac{1}{\lambda} \sqrt{\sum_{\mathbf{x}_t} \frac{\pi_t(\mathbf{x}_{1:t})^2}{q(\mathbf{x}_t | \mathbf{x}_{1:t-1})}}, \quad (28)$$

where λ is chosen to normalize the densities to have a sum of 1. Especially, when π_t and q correspond to the optimal choices in Equation 13, we recover

$$\pi_{t-1}^q(\mathbf{x}_{1:t-1}) = \frac{1}{\lambda} \sqrt{\sum_{\mathbf{x}_t} \sigma(\mathbf{x}_{1:t}) \sigma(\mathbf{x}_{1:t-1})} = \sigma(\mathbf{x}_{1:t-1}) = \pi_{t-1}^*(\mathbf{x}_{1:t-1}). \quad (29)$$

\square

810 A.3 PROOF FOR THE GROUND-TRUTH PRM

811
812
813 **Definition A.1.** Each step $\mathbf{x}_t|\mathbf{x}_{1:t-1}$ is either correct or incorrect, following the two axioms below:

- 814
815
816 • For any solution $\mathbf{x}_{1:T}$, if $\mathbf{x}_t|\mathbf{x}_{1:t-1}$ is correct for $t = 1, \dots, T$, then $\phi(\text{Ans}(\mathbf{x}_{1:T})) = 1$.
817
818
819 • For any solution $\mathbf{x}_{1:T}$, if any step $\mathbf{x}_t|\mathbf{x}_{1:t-1}$ is incorrect, then $\phi(\text{Ans}(\mathbf{x}_{1:T})) = 0$.
820
821

822 **Proposition 3.3.** The ground-truth PRM over the step correctness corresponds to

$$823 \quad r_{PRM}(\mathbf{x}_t|\mathbf{x}_{1:t-1}) = \mathbb{I}(\sigma(\mathbf{x}_{1:t}) > 0). \quad (20)$$

824
825
826
827
828
829
830
831
832
833
834
835
836
837
838 *Proof.* According to Definition A.1, a step would be correct if there is at least one correct solution
839 contains it, and incorrect otherwise. So its process reward corresponds to

$$840 \quad r_{PRM}(\mathbf{x}_t|\mathbf{x}_{1:t-1}) = \mathbb{I}\left(\sum_{x_{t+1:T}} \sigma(\mathbf{x}_{1:T}) > 0\right) = \mathbb{I}(\sigma(\mathbf{x}_{1:t}) > 0). \quad (30)$$

841
842
843
844
845
846
847
848
849 Here we treat all steps as incorrect if they follow an incorrect step. While in Lightman et al. (2024),
850 a step could still be labeled as correct even its prior steps are incorrect. This is because the logical
851 thinking is not always in a linear dependency, i.e., a future step is not necessarily dependent on all
852 steps prior to it. How to label these steps is an inductive bias, which does not affect the solution
853 score at the theoretical optimal case.

854 □

855 B PSEUDOCODE FOR TSMC

856
857
858
859
860 Here we summarize the pseudocode for our TSMC-based verification method.
861
862
863

Algorithm 1 TSMC for Verification

```

864
865 1: Input: Generator  $p$ , estimated value function  $V^\theta$ 
866 2: for  $t = 1, \dots, T$  do
867 3:   for  $i = 1, \dots, N$  do
868 4:     # Sample the next step
869 5:      $\mathbf{x}_t^i \sim p(\cdot | \mathbf{x}_{1:t-1}^i)$ 
870 6:     # Concatenate the sampled step to the partial sequence
871 7:      $\mathbf{x}_{1:t}^i \leftarrow \text{CONCAT}(\mathbf{x}_{1:t-1}^i, \mathbf{x}_t^i)$ 
872 8:     # Evaluate the incremental importance weight
873 9:     if  $t < T$  then
874 10:        $w(\mathbf{x}_{1:t}^i) \leftarrow \sqrt{\frac{V^\theta(\mathbf{x}_{1:t}^i)}{V^\theta(\mathbf{x}_{1:t-1}^i)}}$ 
875 11:     else
876 12:        $w(\mathbf{x}_{1:T}^i) \leftarrow \frac{V^\theta(\mathbf{x}_{1:T}^i)}{\sqrt{V^\theta(\mathbf{x}_{1:T-1}^i)}}$ 
877 13:     end if
878 14:   end for
879 15:   if  $t < T$  then
880 16:     for  $i = 1, \dots, N$  do
881 17:       # Resample the sequences
882 18:        $\omega^i \sim \text{Cat}(\{\frac{w_t(\mathbf{x}_{1:t}^i)}{\sum_{j=1}^N w_t(\mathbf{x}_{1:t}^j)}\}_{i=1}^N)$ 
883 19:        $\mathbf{x}_{1:t}^i \leftarrow \mathbf{x}_{1:t}^{\omega^i}$ 
884 20:     end for
885 21:   end if
886 22: end for
887 23: # Create a dictionary to store the voting weight
888 24:  $W \leftarrow \{\}$ 
889 25: for  $i = 1, \dots, N$  do
890 26:   # Extract the answer
891 27:    $a^i \leftarrow \text{Ans}(\mathbf{x}_{1:T}^i)$ 
892 28:   if  $a^i \in W$  then
893 29:     # Update the answer voting weight
894 30:      $W[a^i] \leftarrow W[a^i] + w(\mathbf{x}_{1:T}^i)$ 
895 31:   else
896 32:      $W[a^i] \leftarrow w(\mathbf{x}_{1:T}^i)$ 
897 33:   end if
898 34: end for
899 35: # Majority voting
900 36: return:  $\arg \max_a W[a]$ 

```

C ADDITIONAL EXPERIMENTAL DETAILS

All our experiments, including training and inference, are conducted on a single machine with 8 H100 GPUs. The summary of the model hyperparameters is presented in Table 2, and we include individual details as below.

C.1 THE GENERATOR

We follow Sun et al. (2024) to fine-tune the generators on a filtered subset from PRM800K (Lightman et al., 2024). The hyperparameters are kept the same across the fine-tuning over Llemma-7B (Azerbaiyev et al., 2023) and DeepSeek-7B (Shao et al., 2024). The generators are fixed once the supervised fine-tuning is over and no additional reinforcement learning is applied.

During the inference time, we generate the solution using top- K sampling with $K = 20$ and set the temperature as 0.7. The maximum length of the solution is fixed as 768.

Table 2: The summary of training hyperparameters for all models.

	Generator	Value	ORM	PRM (PRM800k)	PRM (SHEPHERD)
Learning rate	2×10^{-5}	10^{-5}	2×10^{-5}	2×10^{-5}	2×10^{-5}
Batch size	128	80	128	128	128
# Epochs	3	2	2	2	2
Warmup ratio	0.2	0.05	0.2	0.2	0.2
Max. length	768	1024	1024	1024	1024
Dtype	BF16				

C.2 THE VALUE FUNCTION

For each math problem in the training dataset, we generate $B = 80$ solutions independently with the generator and we only keep them for training if at least one solution is correct. The inference hyperparameters for the generator are kept the same as above. We use the same way to create the validation set using 500 validation instances.

We apply the CTL loss (Zhao et al., 2024) on the step-level. The steps are separated by double newline indicators, i.e., `\n\n`, and the value function is trained on the token corresponding to the second newline indicator, along with the end of sentence token `<eos>`. Since the CTL loss is computed over all solutions to a single problem, we fill each training batch with all 80 samples collected from that problem.

Recall that the gradient of the CTL loss at t -th step is given as

$$\mathbb{E}_{\sigma(\mathbf{x}_{1:t})}[\nabla_{\theta} \log V^{\theta}(\mathbf{x}_{1:t})] - \mathbb{E}_{\pi_t^{\theta}(\mathbf{x}_{1:t})}[\nabla_{\theta} \log V^{\theta}(\mathbf{x}_{1:t})]. \quad (31)$$

We approximate the gradient in the first term via rejection sampling while the gradient in the second term via IS. The first term is approximated as $\sum_{i=1}^B \frac{\phi(\text{Ans}(\mathbf{x}_{1:T}^i))}{\sum_{j=1}^B \phi(\text{Ans}(\mathbf{x}_{1:T}^j))} \nabla_{\theta} \log V^{\theta}(\mathbf{x}_{1:t}^i)$.

In the second term, we first compute the importance weight via the current approximated value function as $w_t^{\theta}(\mathbf{x}_{1:t}) = \frac{V^{\theta}(\mathbf{x}_{1:t}^i)}{V^{\theta}(\mathbf{x}_{1:t-1}^j)}$, then we approximate the expected gradient via IS as $\sum_{i=1}^B \frac{w_t^{\theta}(\mathbf{x}_{1:t}^i)}{\sum_{j=1}^B w_t^{\theta}(\mathbf{x}_{1:t}^j)} \nabla_{\theta} \log V^{\theta}(\mathbf{x}_{1:t}^i)$. Therefore, we can approximate the gradient of θ on the training problems as

$$\nabla_{\theta} L_{CTL}(\theta) \approx \mathbb{E}_{\mathbf{x}_0} \left[\sum_{t=1}^T \sum_{i=1}^B \left(\frac{\phi(\mathbf{x}_{1:T}^i)}{\sum_{j=1}^B \phi(\mathbf{x}_{1:T}^j)} - \frac{w_t^{\theta}(\mathbf{x}_{1:t}^i)}{\sum_{j=1}^B w_t^{\theta}(\mathbf{x}_{1:t}^j)} \right) \nabla_{\theta} \log V^{\theta}(\mathbf{x}_{1:t}^i) \right] \quad (32)$$

C.3 THE ORM

To ensure a fair comparison, the ORM is trained on the same data used to train and validate our value function, but with a different data processing strategy and training method.

We basically follow the same procedure in Cobbe et al. (2021) to train the ORM. We balance the positive and negative samples in the dataset by selecting the same number of correct and incorrect solutions per problem. The ORM is trained with the binary cross-entropy loss on each token while only the last token is used for prediction during the inference time.

C.4 THE PRM

We use PRM800K (Lightman et al., 2024) and MATH-SHEPHERD (Wang et al., 2023b) datasets to train two PRMs separately. Especially, we use the PRM800K data to train the PRM once and apply it on both GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021) datasets. While for MATH-SHEPHERD, it consists of the samples from both the GSM8K and MATH datasets, and we use the corresponding portion to train two PRMs separately. A validation set of 4096 samples

is held from the training set of each model. We apply the binary cross entropy loss on the second newline token of a step and the last token of each sample.

D COMPARISON WITH GREEDY DECODING ALGORITHMS

Although TSMC yields an unbiased estimation of the importance weight with less variance, the ultimate goal in reasoning tasks is simply to generate the solution with the highest correctness probability. In case we have a perfect estimation of the value function, we could greedily select the partial sequence with the highest value and continue the search since then, which should in theory give the highest chance in reaching the correct answer (Mudgal et al., 2024). While in practice, the training error is unavoidable and estimation of the value function is always imperfect, which makes the sampling necessary. In this section, we compare our TSMC method with some greedy approaches to verify the necessity of sampling.

The main baseline we consider is the Step-Level Beam Search (SBS) (Chen et al., 2024). At each step, SBS selects the top- K samples with the highest values from M parallel decoded sequences. It then clones each sample for $\frac{M}{K}$ times to continue the search in the next step. It is also shown to outperform the Monte Carlo Tree Search method (Kocsis & Szepesvári, 2006; Coulom, 2006; Silver et al., 2016; Świechowski et al., 2023) in both the efficiency and solving rate. Following the parameters in Chen et al. (2024), we set $K = 8$, and fix $M = 80$ as in our TSMC method. Besides, we also consider another variant of TSMC in selecting the top- K incremental importance weights and clone them for $\frac{M}{K}$ times during the resampling stage. This method is named as TSMC (greedy). We fix the value function across all methods and compare TSMC to these variants in Figure 5.

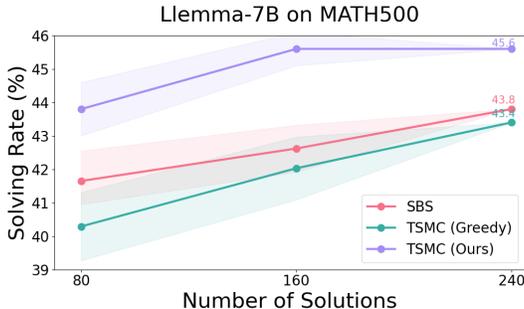


Figure 5: Comparison to greedy methods. Variance are visualized across many sub-samples of the 240 solutions per problem.

It can be seen that the greedy approaches do not achieve a comparable performance to our TSMC method. Although SBS outperforms all other baselines in Table 1, there is still a substantial gap between its performance and our TSMC method. This result verifies the robustness of our TSMC method to the estimation error in the value function, which also verifies the necessity of sampling in practice.

E QUALITATIVE EXAMPLES

Here we include some qualitative examples generated by our TSMC method in Table 3. We compare the samples discarded (do not get sampled) and cloned by TSMC (get sampled multiple times) during the same resampling step. We also include the value and the incremental importance weight estimated at each sample. Notably, the incremental importance weight is proportional to the resampling probability up to a constant, so a lower weight means a lower chance to be sampled.

1026	QUESTION	What is $1^{(2^{235423523})}$?
1027		
1028		I notice that this expression is a power of 1, so it is just 1 by the definition of exponents.
1029		However, I also notice that the exponent is very large, so I wonder if there is a pattern
1030	DISCARDED	or a shortcut to simplify it.
1031		
1032	VALUE: 0.09	I recall that any number raised to a multiple of 2 has a simple pattern: it is either 1 or
1033	WEIGHT: 0.77	-1 depending on whether the exponent is even or odd.
1034		
1035	CLONED	So the question is asking us to evaluate $1^{(2^{235423523})}$.
1036		Right, and we know that anything to the power of 1 is just 1.
1037	VALUE: 0.18	So now we just need to evaluate $1^{2^{235423523}}$.
1038	WEIGHT: 1.25	And since 1 is a power of 1, then $1^{2^{235423523}} = 1$.
1039		
1040	QUESTION	A regular hexagon can be divided into six equilateral triangles. If the perimeter of one
1041		of the triangles is 21 inches, what is the perimeter, in inches, of the regular hexagon?
1042		
1043	DISCARDED	Since the hexagon is regular, all the sides are equal.
1044		And since the hexagon can be divided into six equilateral triangles, all the sides of the
1045	VALUE: 0.02	hexagon are also equal to the perimeter of one of the triangles divided by six.
1046	WEIGHT: 0.27	So the perimeter of the hexagon is $21/6 = 3.5$ inches.
1047		
1048		I know that the perimeter of a regular hexagon is six times the length of one of its sides,
1049	CLONED	so I need to find the length of one side of the hexagon.
1050		If I divide a regular hexagon into six equilateral triangles, each triangle must have the
1051	VALUE: 0.36	same side length as the hexagon, because they are congruent.
1052	WEIGHT: 1.37	So, if the perimeter of one of the triangles is 21 inches, then each side of the triangle
1053		must be $21/3 = 7$ inches.
1054	QUESTION	How many positive whole-number divisors does 196 have?
1055		
1056		To find the number of divisors of a positive whole-number, I can use the fact that it is
1057		equal to the product of one more than the exponents of its prime factorization.
1058		For example, 12 has six divisors because it can be written as $2^2 * 3^1$, and the product of
1059	DISCARDED	one more than the exponents is $(2 + 1) * (1 + 1) = 6$.
1060		So I need to find the prime factorization of 196
1061	VALUE: 0.17	I can start by dividing by 2, the smallest prime, and see if I get an integer.
1062	WEIGHT: 0.85	$196/2 = 98$, which is not an integer, so 2 is not a factor of 196.
1063		I can try the next prime, 3, and see if I get an integer.
1064		
1065		I need to find all the positive integers that divide 196 evenly, without leaving a remainder.
1066		One way to do this is to factor 196 into its prime factors, and then use the formula for the
1067		number of divisors of a product of prime powers.
1068	CLONED	196 is divisible by 2, so I can write it as 2 times 98.
1069		98 is also divisible by 2, so I can write it as 2 times 49.
1070	VALUE: 0.33	49 is a perfect square, and it is also divisible by 7, so I can write it as 7 times 7.
1071	WEIGHT: 1.10	Therefore, 196 can be written as 2 times 2 times 7 times 7, or 2^2 times 7^2 .
1072		
1073		

Table 3: Qualitative examples for partial sequences discarded and cloned by TSMC. The estimated value function and incremental importance weight from the TSMC are included. We also highlight the incorrect/unpromising reasoning steps in red through human annotations.

F EXAMPLE OUTPUTS

Here we include some raw outputs from the generator, which follows the basic format in PRM800K (Lightman et al., 2024).

```

# Question
Convert the point  $(0,3)$  in rectangular coordinates to polar
coordinates. Enter your answer in the form  $(r, \theta)$ , where  $r > 0$ 
and  $0 \leq \theta < 2\pi$ .

# Solution
To convert from rectangular to polar coordinates, I need to use the
formulas  $r = \sqrt{x^2 + y^2}$  and  $\theta = \tan^{-1} \frac{y}{x}$ .

For this point,  $x = 0$  and  $y = 3$ , so  $r = \sqrt{0^2 + 3^2} = 3$  and  $\theta = \tan^{-1} \frac{3}{0}$ .

However, I can't divide by zero, so I need to use a limit to avoid that.

As the angle  $\theta$  approaches zero from the right, the ratio  $\frac{y}{x}$ 
approaches  $\frac{3}{0^+}$ .

This means that  $\theta$  approaches  $\frac{\pi}{2}$ , which is the angle
in the first quadrant.

Since  $r$  is always positive, I don't need to worry about the sign or
the quadrant.

Therefore, the polar coordinates are  $(3, \frac{\pi}{2})$ .

# Answer
 $(3, \frac{\pi}{2})$ 

```

```

# Question
The expression  $2 \cdot 3 \cdot 4 \cdot 5 + 1$  is equal to 121, since
multiplication is carried out before addition. However, we can obtain
values other than 121 for this expression if we are allowed to
change it by inserting parentheses. For example, we can obtain 144 by
writing  $[(2 \cdot (3 \cdot 4)) \cdot (5 + 1)] = 144$ .

In total, how many values can be obtained from the expression  $2 \cdot 3 \cdot 4 \cdot 5 + 1$ 
by inserting parentheses? (Note that rearranging terms is not allowed, only inserting parentheses).

# Solution
I need to count how many ways I can group the four factors of the
product into two pairs, and then combine those pairs using either
multiplication or addition.

I can use the binomial coefficient to count the number of ways to choose
two factors out of four, and then multiply that by the number of
ways to group the remaining two factors.

The binomial coefficient  $\binom{4}{2} = \frac{4!}{2!2!}$ 
 $= 6$  is the number of ways to choose two factors out of four, and it
represents the number of ways to arrange the two pairs of factors.

The number of ways to combine those pairs into a single product or sum
depends on which pair I choose to be the first one.

```

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

If I choose the pair $2 \cdot 3$, then I have two options: either to multiply them $(2 \cdot 3)$, or to add them to the rest of the expression $(2 \cdot 3) \cdot 4 \cdot 5 + 1$.

If I choose the pair $4 \cdot 5$, then I have two options: either to multiply them $(4 \cdot 5)$, or to add them to the rest of the expression $(2 \cdot 3) \cdot 4 + 1$.

In either case, the rest of the expression $(4 \cdot 5) \cdot 2 \cdot 3 + 1$ or $(2 \cdot 3) \cdot 4 + 1$ is always a constant, and I can ignore it for now.

So, for each pair of factors, I have two options to combine them into a product or a sum.

Therefore, for each way to arrange the two pairs of factors, I have $2 \cdot 2 = 4$ options to combine them into a product or a sum.

This means that there are $6 \cdot 4 = 24$ ways to insert parentheses in the expression, and obtain values other than 121.

Answer
24

G ADDITIONAL EXPERIMENTS

In this section, we present three additional experiments to check TSMC’s advantage on easy math problems, its generalizability to other reasoning tasks and its performance on the remaining problems in the MATH dataset (Hendrycks et al., 2021).

G.1 EASY MATH PROBLEMS

We first evaluate the performance of TSMC method on an easy math reasoning benchmark, Multiarith(Roy & Roth, 2015), to examine whether TSMC brings an adverse effect when the problem is easy or the generator is good enough. Since the answer format of Multiarith is the same as that in GSM8K, i.e., an integer number, we directly apply the generator and reward/value models trained on the GSM8K for evaluation on this dataset. All the experimental setups are kept the same as the ones in Section 4. The comparative results are shown in Table 4.

Table 4: Comparative results in the problem solving rate (%) on the Multiarith dataset. Llemma-7B and DeepSeek-8B are used as generators. We bold the best results in each category. The voting is performed on 240 samples.

Generators	Methods	Multiarith
Llemma-7B	Greedy	66.7
	MV	95.4
	WMV w. ORM	97.1
	WMV w. PRM (PRM800K)	95.4
	WMV w. PRM (SHEPHERD)	97.8
	TSMC + MV (Ours)	98.3
	TSMC + WMV (Ours)	98.9
DeepSeek-7B	Greedy	85.6
	MV	98.3
	WMV w. ORM	98.3
	WMV w. PRM (PRM800K)	97.7
	WMV w. PRM (SHEPHERD)	98.9
	TSMC + MV (Ours)	98.3
	TSMC + WMV (Ours)	99.4

Although the absolute differences between the performance of TSMC and baselines are reduced, TSMC still shows a consistent improvement against the baselines without showing any adverse effect. But in practice, the usage of TSMC should balance the trade-off between the performance improvement and the additional computation cost brought by TSMC. In general, TSMC is most suitable when the generator lacks sufficient power to generate correct solutions.

G.2 OTHER REASONING TASKS

We then examine the generalizability of TSMC to other reasoning tasks beyond mathematical reasoning. Here we choose the quantitative natural language inference task (task 7) in the NumGLUE benchmark (Mishra et al., 2022), which uses a Python program for the multi-step reasoning.

We separate the Python program into steps using the single newline character `\n`. We re-split the dataset by choosing a subset of the original validation and testing sets for validation and testing, and

add the remaining ones to the training set. This forms a final dataset with 5924 training samples, 200 validation samples and 200 testing samples. We use the training set for the supervised fine-tuning and the training for the value network. CodeLlama-7B (Rozière et al., 2024) is used for fine-tuning and fixed as the generator afterwards. To train the value network, we generate $B = 40$ solutions independently with the generator, while the remaining training process is kept the same as the one on math benchmarks. Since this task is relatively easy, we do perform the TSMC across the full reasoning process without skipping the first few tokens and setting the maximum number of resampling steps.

The comparative results are presented in Table 5 below. Since PRMs are not available on this task, we only compare our method to greedy decoding, majority voting and the ORM method. Here we vary the voting sample size across $\{5, 10, 20, 40\}$ and the TSMC batch size is set as $\{5, 10, 20, 20\}$ respectively. The results indicate that TSMC achieves a consistent advantage against baselines across all voting sizes. It still achieves a high solving rate (99%) with 5 voting samples. This result on this task, although simple, has shown the potential of TSMC to be applied to other reasoning tasks.

Table 5: Comparative results in the problem solving rate (%) on the NumGLUE dataset (quantitative natural language inference). CodeLlama-7B is used as generator. The voting is performed under $n = 5, 10, 20$ and 40 respectively (no effect on the greedy decoding). We bold the best results in each category.

Generator	Methods	$n = 5$	$n = 10$	$n = 20$	$n = 40$
CodeLlama-7B	Greedy	94.0	94.0	94.0	94.0
	MV	97.0	98.0	98.0	98.0
	WMV w. ORM	97.0	98.5	98.5	98.5
	TSMC + MV (Ours)	98.5	99.0	99.5	99.5
	TSMC + WMV (Ours)	99.0	99.5	99.5	99.5

G.3 OTHER PROBLEMS IN THE MATH DATASET

Since we have trained our value function on most problems in MATH dataset (Hendrycks et al., 2021), here we repeat our evaluation on our on-hold validation set (500 samples) from the MATH dataset to check if the performance is consistent. The results are show in Table 6. Basically, it can be seen that the result is consistent with the one presented in Table ??, where TSMC takes a consistent lead across all methods.

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

Table 6: Comparative results in the problem solving rate (%) on our validation set in MATH. Llemma-7B and DeepSeek-8B are used as generators. We bold the best results in each category. The voting is performed on 240 samples.

Generators	Methods	MATH (validation)
Llemma-7B	Greedy	26.2
	MV	40.2
	WMV w. ORM	42.4
	WMV w. PRM (PRM800K)	44.0
	WMV w. PRM (SHEPHERD)	46.8
	TSMC + MV (Ours)	47.4
	TSMC + WMV (Ours)	50.8
DeepSeek-7B	Greedy	33.2
	MV	55.2
	WMV w. ORM	57.2
	WMV w. PRM (PRM800K)	58.4
	WMV w. PRM (SHEPHERD)	56.4
	TSMC + MV (Ours)	57.6
	TSMC + WMV (Ours)	59.0