FASTER LLM INFERENCE VIA SEQUENTIAL MONTE CARLO

Anonymous authors
Paper under double-blind review

ABSTRACT

Speculative decoding (SD) accelerates language model inference by drafting tokens from a cheap proposal model and verifying them against an expensive target model via rejection sampling. Because rejection truncates the draft block at the first error, throughput degrades when draft and target diverge. Rather than rejecting draft tokens outright, we propose to *reweight* them. To this end, we introduce sequential Monte Carlo speculative decoding (SMC-SD), which replaces token-level rejection with importance-weighted resampling over a population of draft particles. SMC-SD is a principled *approximate* inference scheme that trades exactness for additional speed, while preserving theoretical bounds on its per-step approximation error. Because LLM inference is memory bandwidth-bound, the arithmetic needed to draft particles and to score them in parallel comes nearly for free—SMC-SD uses idle compute to turn verification into a vectorized, fixed-size operation with no rollback. Empirically, SMC-SD achieves $2.43\times$ speed-up over speculative decoding and a $5.2\times$ speed-up over autoregressive decoding, while remaining within 3% of the target model's accuracy on reasoning, instruction-following, and coding benchmarks.

1 INTRODUCTION

Autoregressive generation from neural language models is inherently sequential: each token depends on *all* previous tokens, so generating a single sequence requires as many serial forward passes through the model as there are tokens in the output. This sequential bottleneck is the primary obstacle to faster inference.

Speculative decoding (SD; Leviathan et al., 2023) addresses this bottleneck by amortizing the cost of target-model calls. At its core, SD is a rejection sampler (Chen et al., 2023): a small *draft* model—chosen to be significantly cheaper to evaluate—proposes K-tokens, and the larger *target* model verifies all K tokens in a single forward pass, accepting a prefix and rejecting the rest. This rejection criterion ensures that the resulting strings are distributed *exactly* according to the target, while the speed-up factor is stochastic, depending on the alignment of the draft and target models.

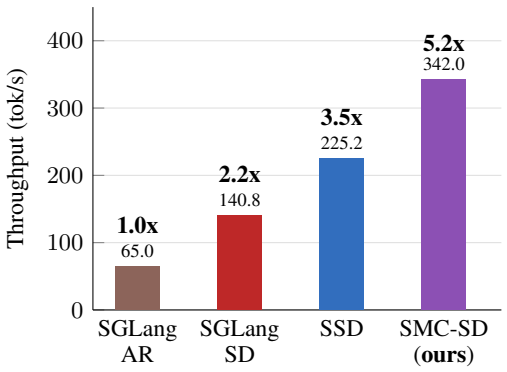


Figure 1: Multi-GPU throughput on ShareGPT for Llama 1B $\rightarrow$ 70B. AR, optimized tree-based SD (SGLang), and SMC-SD use four H100 GPUs; SSD (Kumar et al., 2026) uses five. Bold labels show speed-up over AR.

This paper takes a fundamentally different approach to accelerating autoregressive generation. Rather than searching for a faster exact sampler, we devise an *approximate* sampling scheme whose fidelity to the target can be traded off against speed. The key modification is replacing SD's token-level rejection step with importance-weighted resampling, making the verification step an instance of sequential Monte Carlo (SMC; Doucet et al., 2001; Del Moral

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

et al., 2006; Naesseth et al., 2024). Our method *reverses* the guarantee of SD—approximation quality is stochastic and the speed-up factor is deterministic. We call our method **sequential Monte Carlo speculative decoding (SMC-SD)** and describe it concretely in § 3.

We derive a closed-form expression for the tokens-per-second (TPS) speed-up factor of SMC-SD with the roofline model (Williams et al., 2009) of the GPU in § 3.1. We show that the roofline model gives a simple characterization of the speed-up factor in both the memory bandwidth-bound and compute-bound regimes.

From a theory perspective, because SMC-SD is ultimately based on importance resampling, non-asymptotic bounds on its approximation error can be obtained using known techniques. For the SMC-SD variant with one multinomial resampling step per round, we show that the mean squared error over a fixed generation horizon decays as $O(1/N)$, with constants governed by the per-round χ^2-divergence between target and draft conditionals (§ 3.2).

From a systems perspective, SMC-SD presents a novel computation pattern that necessitates designing an inference engine around it. In § 3.3, we use several observations at the intersection of hardware and algorithm to guide the design of our high-throughput SMC-SD inference engine. Both the drafting and verifying stages of SMC-SD use a vectorizable execution structure that map naturally onto GPU hardware and enable greater parallelism than standard speculative decoding. For KV cache management, we observe that all KV cache data movement during the resampling step can be replaced with efficient in-place pointer exchanges, and we reduce the size of the cache by leveraging the shared prefix among particles.

Empirically, SMC-SD achieves up to $2.5\times$ the throughput of an optimized implementation of speculative decoding on GSM8K (Cobbe et al., 2021), MATH500 (Lightman et al., 2023), AlpacaEval (Dubois et al., 2025), and DS1000 (Lai et al., 2022) across both Llama (Team, 2024) and Qwen (Yang et al., 2025) model families, while remaining within 3% of the target model's accuracy. In a multi-GPU setting, SMC-SD achieves a $2.43\times$ speed-up over optimized tree-based speculative decoding (SGLang SD) and $5.2\times$ over the autoregressive baseline. Figure 1 summarizes these end-to-end throughput gains.

2 Language Models and Speculative Decoding

2.1 Language Modeling Background

Let $\mathcal{X}$ be a finite **alphabet**, a finite, non-empty set. We call elements of $\mathcal{X}$ **tokens**. We write $\mathcal{X}^*$ for the set of all finite strings over $\mathcal{X}$, including the empty string ε, and denote individual tokens by x and strings by $\boldsymbol{x} = x_1 \cdots x_T \in \mathcal{X}^*$. We use the shorthand $\boldsymbol{x}_{<t} \stackrel{\text{def}}{=} (x_1, \ldots, x_{t-1})$ to denote the prefix up to but not including position t. A **language model** is a probability distribution p over $\mathcal{X}^*$. Given a language model p over $\mathcal{X}^*$, we define the **prefix probability** of a string $\boldsymbol{x}$ as the total probability of all strings that begin with $\boldsymbol{x}$ as follows $\overrightarrow{p}(\boldsymbol{x}) \stackrel{\text{def}}{=} \sum_{\boldsymbol{x}' \in \mathcal{X}^*} p(\boldsymbol{x}\boldsymbol{x}')$. The ratio of consecutive prefix probabilities recovers a conditional distribution over the next token $x \in \mathcal{X} \cup \{\text{EOS}\}$ where $\text{EOS} \notin \mathcal{X}$ is a distinguished symbol:

$$\overrightarrow{p}(y \mid \boldsymbol{x}) = \frac{\overrightarrow{p}(\boldsymbol{x}\, y)}{\overrightarrow{p}(\boldsymbol{x})} \qquad \text{(1a)} \qquad\qquad \overrightarrow{p}(\text{EOS} \mid \boldsymbol{x}) = \frac{p(\boldsymbol{x})}{\overrightarrow{p}(\boldsymbol{x})} \qquad \text{(1b)}$$

Every language model admits an **autoregressive factorization**: $p(\boldsymbol{x}) = \overrightarrow{p}(\text{EOS} \mid \boldsymbol{x}) \prod_{t=1}^{T} \overrightarrow{p}(x_t \mid \boldsymbol{x}_{<t})$. We call a language model that is autoregressively factorized an **autoregressive language model**. Finally, we say that a language model p is **absolutely continuous** with respect to a second language model q, written $p \ll q$, if $q(\boldsymbol{x}) = 0 \implies p(\boldsymbol{x}) = 0, \forall \boldsymbol{x}$.

2.2 Speculative Decoding

Speculative decoding (Leviathan et al., 2023; Chen et al., 2023) accelerates autoregressive inference by applying **rejection sampling** at the token level. We consider a **target model** p and a smaller, faster-to-evaluate **draft model** q over the same alphabet $\mathcal{X}$. Given a draft length K and the draft–target pair (q, p), each round proceeds in two phases. In the **draft** phase, the draft model generates a continuation $\boldsymbol{d}$ of up to K tokens autoregressively from $\overrightarrow{q}$. In the **verification** phase, the target

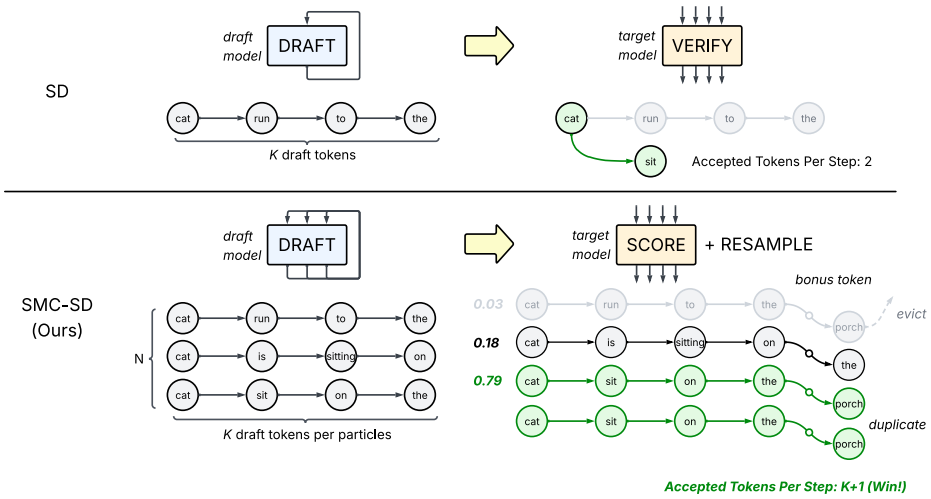


Figure 2: Our approach (bottom) compared to standard speculative decoding (top). In standard speculative decoding, a draft model generates a single sequence of draft tokens. A target model then performs a verification step, accepting a valid prefix of verified tokens while discarding rejected tokens. SMC-SD maintains a set of N candidate sequences (particles). In each iteration, the draft model extends the N sequences by K draft tokens, where K equals 4 in the illustration above. The target model then scores these extensions and samples a bonus token. In the Resample phase, sequences with low score (e.g., .03) are evicted, while sequences with high probabilities (e.g., .79) are duplicated to form the new set of N sequences.

model scores all K draft tokens in a single forward pass. Each token d_j is accepted with probability $\alpha_j = \min\big(1,\ \overrightarrow{p}(d_j \mid \boldsymbol{x}\, \boldsymbol{d}_{<j}) / \overrightarrow{q}(d_j \mid \boldsymbol{x}\, \boldsymbol{d}_{<j})\big)$. On the first rejection, the token is resampled from the residual distribution and all subsequent draft tokens are discarded; if every token is accepted, a bonus token is drawn from $\overrightarrow{p}(\cdot \mid \boldsymbol{x}\, \boldsymbol{d})$. In either case, every round produces at least one token whose distribution involves the target model. This procedure guarantees that the output is distributed *exactly* according to p (Leviathan et al., 2023); the full pseudocode is given in § A. The effective speed-up depends on the *acceptance rate*: the closer q is to p, the more tokens survive per round.

3 Sequential Monte Carlo Speculative Decoding

Sequential Monte Carlo (SMC; Doucet et al., 2001; Del Moral et al., 2006; Naesseth et al., 2024) runs a population of N particles $\{\boldsymbol{x}^{(n)}\}_{n=1}^{N}$; each particle is a string that we intend to keep extending. Each particle alternates between three operations at each step t:

1. **Extend:** sample $x_t^{(n)} \sim \overrightarrow{q}(\cdot \mid \boldsymbol{x}^{(n)})$ for each particle, appending the token to $\boldsymbol{x}^{(n)}$.
2. **Reweight:** update each weight by $w_t^{(n)} \leftarrow w_{t-1}^{(n)} \cdot \overrightarrow{p}(x_t^{(n)} \mid \boldsymbol{x}_{<t}^{(n)}) / \overrightarrow{q}(x_t^{(n)} \mid \boldsymbol{x}_{<t}^{(n)})$.
3. **Resample:** if the effective sample size (ESS) falls below a threshold, draw new ancestor indices $a_1, \ldots, a_N$ independently from a categorical distribution over $\{1, \ldots, N\}$ with probabilities proportional to the normalized weights $\overline{w}^{(1)}, \ldots, \overline{w}^{(N)}$, set $\boldsymbol{x}^{(n)} \leftarrow \boldsymbol{x}^{(a_n)}$ for all n, and reset all weights to $1/N$.

Resampling concentrates computation on promising prefixes and prevents weight degeneracy from rendering the particle population uninformative. The **effective sample size** (ESS) (Huggins, 2014) quantifies how many particles contribute meaningfully:

$$\mathrm{ESS} \;=\; \frac{\big(\sum_{n=1}^{N} w^{(n)}\big)^2}{\sum_{n=1}^{N} \big(w^{(n)}\big)^2}. \tag{2}$$

The ESS satisfies $1 \le \mathrm{ESS} \le N$: when $\mathrm{ESS} \approx N$, all particles contribute meaningfully; when ESS collapses toward 1 (**weight degeneracy**), a single particle dominates. The *resample* step in SMC is triggered when ESS falls below a threshold, pruning low-weight particles and duplicating high-weight ones to maintain sample diversity.

Algorithm 1 Sequential Monte Carlo Speculative Decoding (One Round)

Require: Particles $\{\boldsymbol{x}^{(n)}\}_{n=1}^{N}$, draft model q, target model p, draft length K, ESS threshold $\eta \in [0, N]$
Ensure: Updated particles $\{(\boldsymbol{x}^{(n)}, w^{(n)})\}_{n=1}^{N}$, each extended by $K{+}1$ tokens (K draft + 1 bonus)

— Draft —
for $n = 1, \ldots, N$ **do** ▷ sample N draft extensions in parallel
  $\boldsymbol{d}^{(n)} \leftarrow \varepsilon$
  for $t = 1, \ldots, K$ **do**
    $x_t \sim \overrightarrow{q}(\cdot \mid \boldsymbol{x}^{(n)}\, \boldsymbol{d}^{(n)})$
    $\boldsymbol{d}^{(n)} \leftarrow \boldsymbol{d}^{(n)} x_t$
  end for
end for
— Score and bonus token —
for $n = 1, \ldots, N$ **do** ▷ single batched target forward pass
  Compute $\overrightarrow{p}(d_j^{(n)} \mid \boldsymbol{x}^{(n)}\, \boldsymbol{d}_{<j}^{(n)})$ for all $j \in \{1, \ldots, K\}$
  $x_n^{+} \sim \overrightarrow{p}(\cdot \mid \boldsymbol{x}^{(n)}\, \boldsymbol{d}^{(n)})$ ▷ bonus token
end for
— Reweight —
for $n = 1, \ldots, N$ **do**
  $w^{(n)} \leftarrow w^{(n)} \cdot \prod_{j=1}^{K} \dfrac{\overrightarrow{p}(d_j^{(n)} \mid \boldsymbol{x}^{(n)}\, \boldsymbol{d}_{<j}^{(n)})}{\overrightarrow{q}(d_j^{(n)} \mid \boldsymbol{x}^{(n)}\, \boldsymbol{d}_{<j}^{(n)})}$ ▷ incremental block weight
end for
$\overline{w}^{(n)} \leftarrow w^{(n)} / \sum_{m=1}^{N} w^{(m)}$ for all n ▷ normalize
$\mathrm{ESS} \leftarrow \dfrac{\left(\sum_{n=1}^{N} w^{(n)}\right)^2}{\sum_{n=1}^{N} (w^{(n)})^2}$ ▷ effective sample size
— Resample —
if $\mathrm{ESS} < \eta$ **then**
  for $n = 1, \ldots, N$ **do**
    Sample ancestor $a_n \sim \mathrm{Cat}(\overline{w}^{(1)}, \ldots, \overline{w}^{(N)})$ ▷ $\Pr[a_n = m] = \overline{w}^{(m)}$
  end for
  $\boldsymbol{x}^{(n)} \leftarrow \boldsymbol{x}^{(a_n)}\, \boldsymbol{d}^{(a_n)}\, x_{a_n}^{+}$ for all $n \in [N]$ ▷ duplicate high-weight particles
  $w^{(n)} \leftarrow 1/N$ for all $n \in [N]$ ▷ reset weights after resampling
else
  $\boldsymbol{x}^{(n)} \leftarrow \boldsymbol{x}^{(n)}\, \boldsymbol{d}^{(n)}\, x_n^{+}$ for all $n \in [N]$ ▷ keep existing ancestry
end if
return $\{(\boldsymbol{x}^{(n)}, w^{(n)})\}_{n=1}^{N}$ ▷ N updated particles, each extended by $K{+}1$ tokens

The SMC-SD Algorithm. We instantiate the SMC framework for speculative decoding in Algorithm 1: the target is p, the proposal is a cheaper draft model q, and one **round** extends every particle by a draft of up to K tokens. At the start of each round, the particle population represents a set of partially generated strings $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(N)}$; particles may share ancestry through earlier resampling steps, but they need not share the same current prefix. For a weighted particle population $\{(\boldsymbol{x}^{(n)}, w^{(n)})\}_{n=1}^{N}$, draft length K, and ESS threshold $\eta \in [0, N]$, a single round proceeds as in Algorithm 1. Unlike SD, no token-level accept/reject chain truncates a draft midway; every particle is extended by exactly K tokens per round (Figure 2).

Each particle independently drafts K sequential tokens from its own current prefix under q, then the target model p scores all N sequences in a single batched forward pass and generates a bonus token. The resulting block weights update the particle population, and a low-ESS resampling step duplicates high-weight trajectories while pruning low-weight ones. The round updates the internal particle population block by block; after the final round, one complete sequence is sampled from the terminal normalized weights and returned as the model output. That is, SMC-SD maintains latent particle trajectories during generation and samples the returned sequence only at termination.

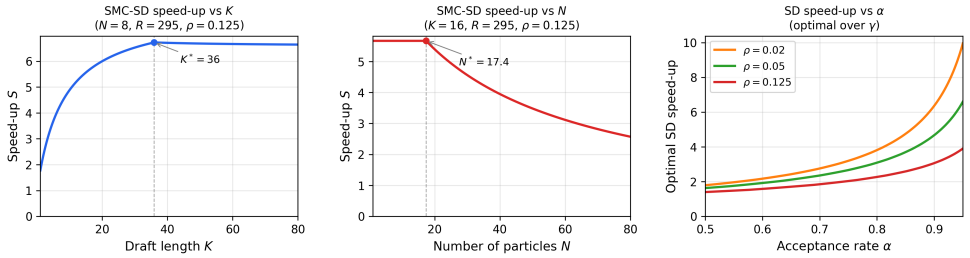


Figure 3: **Left, middle:** Theoretical speed-up of SMC-SD over autoregressive decoding for a Llama-1B $\rightarrow$ 8B pair as a function of draft length K (left) and number of particles N (middle), with $\rho = 1/8$, $B = 1$, and $R = 295$. Dashed lines mark the ridge point where the target forward pass transitions from memory-bound to compute-bound. In the memory-bound regime (left of the ridge point), increasing K increases the speed-up factor and adding particles is free; past the ridge point, speed-up is approximately constant as K increases, and it degrades linearly as N grows. **Right:** optimal speculative decoding speed-up from Leviathan et al. (2023), $(1 - \alpha^{\gamma+1})/((1 - \alpha)(\gamma c + 1))$, as a function of acceptance rate α after optimizing over γ (i.e. K), shown for several draft-to-target cost ratios ρ.

3.1 Speed-up Analysis

We analyze SMC-SD TPS under the roofline model (Williams et al., 2009), which classifies a workload according to its *arithmetic intensity*, or the floating-point operations (FLOPs) used per byte of memory loaded for the workload. Whether a workload is memory bandwidth-bound or compute-bound depends on whether its arithmetic intensity is greater than or less than the *ridge point* of the hardware, or its peak FLOPs per memory bandwidth. Let B denote the number of independent user requests served concurrently (the *batch size*), and let the *effective batch size* denote BN, the total number of particles across all requests. The target model's forward pass processes $BN(K+1)$ tokens per round: N particles per request, each extended by K draft tokens plus one bonus token. Let $c = T_q/T_p \ll 1$ be the draft-to-target decoding-time ratio; under the roofline model, $c \approx \rho \stackrel{\text{def}}{=} \mathcal{P}_q/\mathcal{P}_p$ (assuming the draft and target have the same quantization), since memory-bound latency is proportional to parameter count. Let R denote the effective batch size (in tokens) at which a forward pass reaches the ridge point; this depends only on the hardware and quantization format. Below, we characterize the speed-up factor depending on whether the verification pass is memory-bound or compute-bound (proofs in Theorems G.3 and G.4).

Memory-bound regime. When $BN(K+1) \leq R$, the target scores all $BN(K+1)$ tokens in a single weight-load and the speed-up over autoregressive decoding is

$$S^{(\text{mem})} = \frac{K+1}{\rho K + 1}, \tag{3}$$

independent of BN—particles are "free."

Compute-bound regime. Once $BN(K+1)$ exceeds R, the target pass becomes compute-bound. Writing $\hat{\rho}(BN) = \max(\rho, \rho BN/R)$ for the effective draft cost ratio, the speed-up falls to

$$S^{(\text{comp})} = \frac{K+1}{\hat{\rho}(BN)\, K + BN(K+1)/R}, \tag{4}$$

which degrades linearly in the effective batch size after the ridge point has been reached.

Figure 3 shows the theoretical speed-up for multiple values of N and K. The figure uses a Llama-1B $\rightarrow$ 8B pair. The speed-up factor increases with respect to K up until the ridge point is reached; at which point, the speed-up becomes approximately constant. The speed-up factor is constant with respect to N in the memory-bandwidth bound regime, but degrades linearly once the workload becomes compute-bound.

3.2 Characterizing SMC-SD Approximation Error

Because SMC-SD is an approximate sampler, it is important to characterize its approximation quality. Fortunately, since SMC-SD is based on sequential importance sampling with resampling, non-asymptotic bounds on its divergence from a target distribution can be obtained using known techniques (Agapiou et al., 2015; Del Moral et al., 2006; Naesseth et al., 2024). We establish a non-asymptotic bound on the mean squared error of SMC-SD as a function of the number of particles N.

SMC-SD proceeds over rounds $t = 0, 1, \ldots, T$, each extending every particle by one block of up to $K{+}1$ tokens. Let $\mathcal{X}_t^* \subseteq \mathcal{X}^*$ be the finite set of prefixes reachable after round t. The particle population after round t defines the weighted empirical measure

$$\widehat{p}_t^{\,N} \stackrel{\text{def}}{=} \sum_{n=1}^{N} \overline{w}_t^{(n)} \, \delta_{\boldsymbol{x}_t^{(n)}}, \qquad \overline{w}_t^{(n)} \stackrel{\text{def}}{=} \frac{w_t^{(n)}}{\sum_{m=1}^{N} w_t^{(m)}}, \qquad w_t^{(n)} \stackrel{\text{def}}{=} \frac{p(\boldsymbol{x}_t^{(n)} \mid \boldsymbol{x}_{t-1}^{(n)})}{q(\boldsymbol{x}_t^{(n)} \mid \boldsymbol{x}_{t-1}^{(n)})}, \tag{5}$$

the random distribution from which SMC-SD draws its output at the terminal round $t = T$; the $w_t^{(n)}$ are the block weights of Algorithm 1 and $\overline{w}_t^{(n)}$ their normalization. As $N \to \infty$, $\widehat{p}_t^{\,N}$ converges to p_t almost surely (Del Moral et al., 2006). For finite N, however, $\widehat{p}_t^{\,N}$ is a *random* distribution that depends on the particular draw of proposals and resampling decisions.

The difficulty of round t is captured by the local chi-square divergence between the target and draft conditionals at the parent prefix,

$$\chi_t^2(\boldsymbol{x}') \stackrel{\text{def}}{=} \chi^2\big(p(\cdot \mid \boldsymbol{x}') \,\big\|\, q(\cdot \mid \boldsymbol{x}')\big) = \mathop{\mathbb{E}}_{\boldsymbol{x} \sim q(\cdot \mid \boldsymbol{x}')}\left[\left(\tfrac{p(\boldsymbol{x} \mid \boldsymbol{x}')}{q(\boldsymbol{x} \mid \boldsymbol{x}')}\right)^2\right] - 1, \qquad \bar{\chi}_t^2 \stackrel{\text{def}}{=} \sup_{\boldsymbol{x}' \in \mathcal{X}_{t-1}^*} \chi_t^2(\boldsymbol{x}'), \tag{6}$$

the per-round analogue of the single-step divergence $\chi^2(p\|q)$.

Theorem 3.1 (SMC-SD MSE bound)**.** *Consider the SMC-SD variant that performs one multinomial resampling step at the start of each round (§ B). If the per-round divergences are uniformly bounded, $\bar{\chi}_t^2 \le \bar{\chi}^2$ for all t, then for a fixed generation horizon T,*

$$\mathbb{E}\big[\|\widehat{p}_T^{\,N} - p_T\|_2^2\big] \;\lesssim_T\; \frac{1 + \bar{\chi}^2}{N}. \tag{7}$$

Theorem 3.1 (proof and supporting lemmata in § B) bounds the error of the full multi-round SMC-SD process: for a fixed horizon the mean squared error decays as $O(1/N)$ in the number of particles, with the constant governed by the per-round χ^2-divergence between the target and draft conditionals. This gives practitioners a concrete lever—increasing N tightens the approximation at every round—which we validate empirically in § 4, where these guarantees translate into strong end-to-end accuracy across benchmarks.

3.3 High-Throughput SMC-SD Engine Design

SMC-SD is not merely an algorithm that happens to run on GPUs—its vectorized structure maps onto the batch-parallel execution model of modern accelerators and makes use of modern optimizations on inference engines including dynamic batching, PagedAttention, and RadixAttention (Yu et al., 2022; Kwon et al., 2023; Zheng et al., 2024). As such, we build our SMC-SD engine as a fork of SGLang (Zheng et al., 2024). We describe our high-throughput inference engine for SMC-SD, organized around a series of observations at the intersection of algorithm and hardware.

Observation 1: SMC-SD converts under-utilized compute into speed. Autoregressive decoding is heavily memory bandwidth-bound: each token requires loading the full model weights yet performs only a single vector–matrix multiply. The *arithmetic intensity*, the FLOPs performed per byte of memory load, for a forward pass of a language model quantized to b bytes per parameter remains at just $\mathrm{AI}_{\mathrm{AR}} = 2/b$. Standard speculative decoding increases the arithmetic intensity by a factor of K but both methods remain far below the roofline ridge points of modern GPUs—312 for A100, 412 for H200, 584 for B300 in FP16 (NVIDIA Corporation, 2020; 2024b;c;a)—leaving the vast majority of available compute idle.

SMC-SD closes this gap by leveraging parallelism along the *batch-wise* and *sequence-wise* axes. The draft model uses dynamic batching to concurrently generate N tokens at a time in lockstep, followed by the target model scoring a block of $N(K+1)$ tokens using a single re-prefill step. Let $\mathcal{P}$ denote the target model's parameter count, b the bytes per parameter (b=2 for FP16, b=1 for INT8), and $\rho = \mathcal{P}_q/\mathcal{P}_p$ the draft-to-target parameter ratio. Over a full SMC-SD cycle, the arithmetic intensity is (proofs in § G)

$$\text{AI}_{\text{cycle}} = \frac{2N(\rho K+K+1)}{b(\rho K+1)} \approx \frac{2N(K+1)}{b} \quad \text{when } \rho \ll 1, \tag{8}$$

an $\approx N\times$ increase over standard SD. For Llama-3.1-70B in FP16 with N=8 and K=16, $\text{AI}_{\text{cycle}} \approx$ 136 compared to just 17 for standard SD with the same K.

Observation 2: Pointer operations turn prefix-sharing into compute and memory gains. The resampling step in SMC-SD introduces a novel challenge in inference engine design. When resampling duplicates a high-weight particle, a naïve implementation would require KV-cache data movement across particles and *re-prefill* for the duplicated sequences. In SMC-SD, we avoid both of these costs by exploiting the prefix-sharing structure together with page-based KV-cache management (Kwon et al., 2023) and RadixAttention (Zheng et al., 2024). Specifically, duplicated particles share exactly the same prefix as their ancestors up to the resampling point, so the engine can perform resampling by copying page metadata and incrementing the reference counts rather than copying the underlying KV tensors directly. As a result, resampling is implemented as an $O(\ell_{\text{seq}})$ page-metadata operation, rather than an $O(\ell_{\text{seq}} \times d_{\text{model}})$ KV-tensor copy. Exploiting prefix sharing not only avoids KV-cache data movement and reduces FLOPs, but also lowers memory usage. Empirically, for the Llama-1B $\rightarrow$ 8B, N=8, K=16 configuration, our optimization yields a **72.3%** reduction in KV-cache usage.

4 Experiments

We characterize the *speed–accuracy Pareto frontier* of SMC-SD on a single H100 GPU across GSM8K, MATH500, AlpacaEval, and DS1000 (Dubois et al., 2025; Cobbe et al., 2021; Lai et al., 2022; Lightman et al., 2023). We test two model families—Llama 3.2-1B$\rightarrow$3.1-8B, and Qwen 0.5B$\rightarrow$14B (SD) versus Qwen 3B$\rightarrow$14B (SMC-SD)[1]—to demonstrate that SMC-SD's gains are consistent across architectures and draft-to-target size ratios.

GSM8K, MATH500, AlpacaEval, and DS1000. Figure 4 shows the speed–accuracy Pareto frontier for SMC-SD and optimized SD on GSM8K, MATH500, AlpacaEval, and DS1000. Across all four benchmarks and both model families, the SMC-SD frontier lies above and to the right of the SD baseline, indicating simultaneous improvements in throughput and accuracy or favorable tradeoffs between the two. Table 1 quantifies the gains at two operating points: *iso-accuracy* (the fastest SMC-SD configuration within 3 percentage points of SD's accuracy) and *max speed* (within 10 percentage points). At iso-accuracy, SMC-SD achieves 1.1–2.5$\times$ the throughput of SD; allowing a modest accuracy relaxation widens this to up to 3.4$\times$ (§ C).

Comparison against recent speculative decoders. § F reports a further comparison on a Qwen3-14B target against DSpark (DeepSeek-AI, 2026) and EAGLE3, using the public draft checkpoints trained for that model. SMC-SD attains 1.95$\times$ the throughput of DSpark at batch size 1, but this margin decays monotonically with batch size and inverts by batch size 16, where DSpark is both faster and exact. We report the full sweep, including the crossover, since it delimits the regime in which trading exactness for throughput is profitable.

Despite being an approximate sampler, SMC-SD maintains strong downstream accuracy: every point on the Pareto frontier in Figure 4 remains within 15% of the target model's accuracy. The throughput gains come from a structural difference in how SMC-SD uses the target forward pass. In standard SD, the first rejected token truncates the draft and triggers a KV-cache rollback, so the number of

[1]For Qwen, the 0.5B draft makes for a faster speculative decoder than 3B because the smaller draft is faster and SD's acceptance rates do not justify the larger model. SMC-SD, by contrast, benefits from the stronger 3B proposal, so we compare each method with its best draft model.

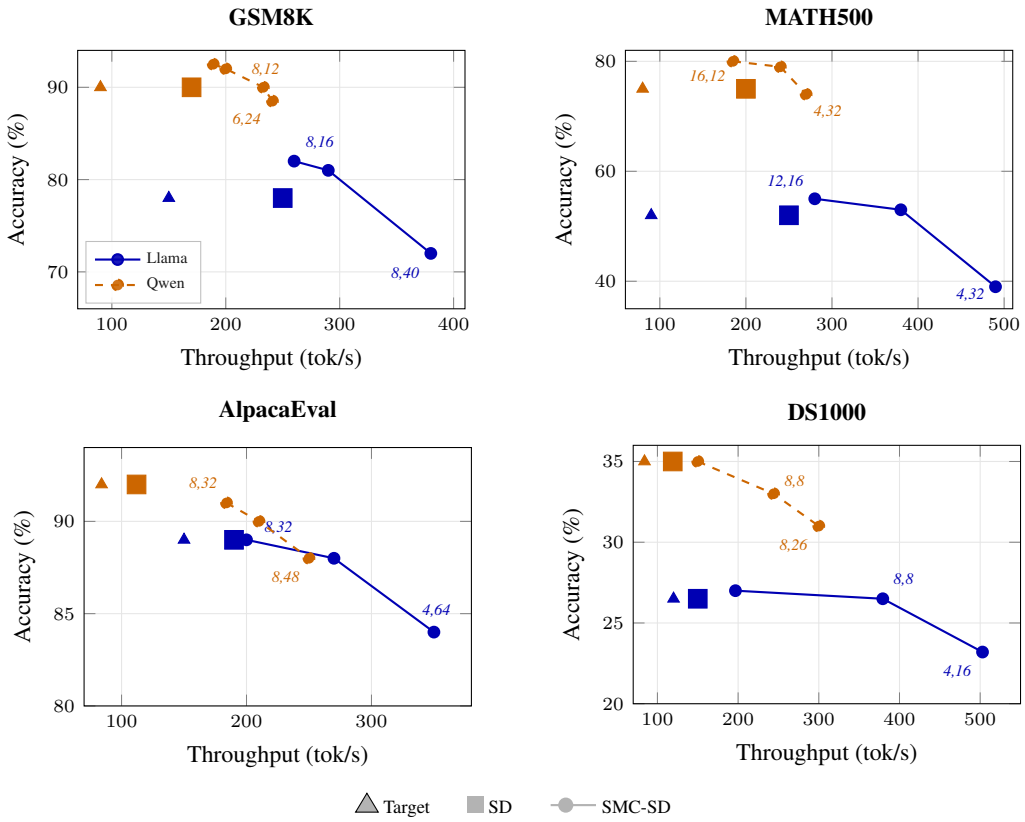


Figure 4: Speed–accuracy Pareto frontier on GSM8K (top left), MATH500 (top right), AlpacaEval (bottom left), and DS1000 (bottom right). Blue/solid: Llama 3.2-1B $\rightarrow$ 3.1-8B; orange/dashed: Qwen (SD: 0.5B$\rightarrow$14B; SMC-SD: 3B$\rightarrow$14B). Italic labels show (N, K) configurations along the SMC-SD Pareto frontier. Experiments were conducted using a single H100 SXM GPU.

tokens produced per round is variable and often much less than $K{+}1$. SMC-SD always produces exactly $K{+}1$ tokens per particle per round—there is nothing to roll back.

Multi-GPU throughput. We deploy SMC-SD on a Llama 3.2-1B $\rightarrow$ Llama 3-70B draft-target pair across 4 H100 GPUs. Figure 1 summarizes the results: SMC-SD achieves 342 tok/s, a $5.2\times$ speed-up over the autoregressive baseline and a $2.43\times$ speed-up over optimized tree-based speculative decoding (SGLang SD). Figure 5 shows throughput per GPU as a function of batch size for both the 1B$\rightarrow$8B and 1B$\rightarrow$70B pairs; SMC-SD leads across most batches tested.

Comparison to lossy tree speculative decoding. We compare SMC-SD on ShareGPT at batch size 1 with SGLang's relaxed-acceptance tree SD, which recovers exact tree verification at threshold $\delta{=}1$, and with Cactus (Hao & Mou, 2026), a KL-budgeted acceptance rule in its native vLLM implementation. SMC-SD outperforms the best lossy tree configuration by $1.54\times$ (1B$\rightarrow$8B) and $1.97\times$ (1B$\rightarrow$70B), and Cactus by $1.73\times$ and $2.57\times$. Prefill is comparable to chain SD, while shared particle ancestry limits peak KV occupancy to $1.45\times$ the chain-SD baseline rather than the naive $N{=}6\times$. Full proposer details, configurations, overheads, and measurement protocols are given in § E.

5 Related Work

SMC for Language Models. Recent work has applied SMC to large language models for *constrained generation*: steering outputs toward syntactic constraints (Loula et al., 2025; Xefteri et al., 2025), semantic specifications (Lew et al., 2023), or reward-defined posteriors (Zhao et al., 2024).

In these settings, the target is a posterior $p(x \mid \text{constraint})$ and the proposal is the unconstrained language model. SMC-SD repurposes the same machinery for *inference acceleration*: the target is p itself and the proposal is a faster draft model q.

Exact speculative decoding. Tree decoding algorithms (Cai et al., 2024; Li et al., 2025b) extend standard SD by proposing a tree of tokens at each position, increasing the expected number of accepted tokens while preserving exact sampling. SMC-SD trades exactness for speed via a tunable approximation.

Approximate speculative decoding. Several methods relax exactness through heuristic acceptance criteria. FSD (Holsman et al., 2025) accepts tokens based on draft–target divergence; FLy (Li et al., 2025a) uses an entropy gate with a deferred acceptance window; Judge Decoding (Bachmann et al., 2025) trains a judge network to replace rejection sampling. All retain a per-token accept/reject gate: a single rejection truncates the block, each token's contribution is binary, and outputs are variable-length. SMC-SD instead assigns each *block* a continuous importance weight, produces a fixed number of tokens per round, and provides asymptotic convergence guarantees—all without additional training.

Complementary strategies. Several orthogonal techniques can be combined with SMC-SD. On the *drafting* side, EAGLE (Li et al., 2024; 2025b;c) conditions the draft model on target-model embeddings, Medusa (Cai et al., 2024) attaches parallel decoding heads, and DFlash (Chen et al., 2026) and DSpark (DeepSeek-AI, 2026) emit whole draft blocks in parallel, the latter via a semi-autoregressive backbone—all producing higher-quality or faster proposals that would directly benefit SMC-SD. On the *scheduling* side, Specdec++ (Huang et al., 2025) adaptively modulates speculation length, SSD (Kumar et al., 2026) asynchronously overlaps drafting and verifying, and DSpark's confidence scheduler converts per-block survival into a per-request verification budget; analogous adaptive strategies over N and K are a natural extension of SMC-SD. All results use the optimal temperature parameters for each task for both SMC-SD and SD.

6 Conclusion

We presented SMC-SD, an algorithm that accelerates LLM inference by replacing speculative decoding's token-level rejection with importance-weighted resampling over a population of draft particles. We established a non-asymptotic error bound showing that the multi-round MSE of the variant with one multinomial resampling step per round decays as $O(1/N)$ for a fixed generation horizon. Empirically, SMC-SD achieves $2.43\times$ the throughput of optimized speculative decoding and $5.2\times$ that of autoregressive decoding in a multi-GPU setting, while remaining within 3% of target-model accuracy across reasoning, instruction-following, and coding benchmarks.

The same importance-weighting machinery can also target distributions proportional to $p(x)\Psi(x)$, enabling sequence-level power sampling, constrained generation, and reward-weighted decoding; § I demonstrates the first. Future work includes overlapping resampling with drafting, disaggregating draft and target computation, and adapting N and K dynamically to hardware and workload characteristics.

AI use statement

In this work, we used generative AI tools to assist with implementing portions of the SMC-SD software and to review the presentation and internal consistency of the methodology, experiments, and mathematical arguments. We additionally used generative AI tools to edit the manuscript for clarity and concision and to revise LaTeX formatting, figures, captions, cross-references, and bibliography entries. We did not use generative AI to generate experimental data or conduct experiments. All AI-assisted code and manuscript revisions were reviewed by the authors; the code was tested, experiments were run by the authors, and reported values were checked against the underlying results. We take responsibility for the final content of this work, including all AI-assisted text, claims, code, and artifacts.

REFERENCES

Sergios Agapiou, Omiros Papaspiliopoulos, Daniel Sanz-Alonso, and Andrew M. Stuart. Importance sampling: Intrinsic dimension and computational cost. *Statistical Science*, 32:405–431, 2015. URL https://projecteuclid.org/journals/statistical-science/volume-32/issue-3/Importance-Sampling-Intrinsic-Dimension-and-Computational-Cost/10.1214/17-STS611.pdf.

Seyedarmin Azizi, Erfan Baghaei Potraghloo, Minoo Ahmadi, Souvik Kundu, and Massoud Pedram. Power-smc: Low-latency sequence-level power sampling for training-free llm reasoning, 2026. URL https://arxiv.org/abs/2602.10273.

Gregor Bachmann, Sotiris Anagnostidis, Albert Pumarola, Markos Georgopoulos, Artsiom Sanakoyeu, Yuming Du, Edgar Schönfeld, Ali Thabet, and Jonas Kohler. Judge decoding: Faster speculative sampling requires going beyond model alignment. *arXiv preprint arXiv:2501.19309*, 2025. URL https://arxiv.org/abs/2501.19309.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. Medusa: Simple llm inference acceleration framework with multiple decoding heads, 2024. URL https://arxiv.org/abs/2401.10774.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling, 2023. URL https://arxiv.org/abs/2302.01318.

Jian Chen, Yesheng Liang, and Zhijian Liu. Dflash: Block diffusion for flash speculative decoding, 2026. URL https://arxiv.org/abs/2602.06036.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

DeepSeek-AI. Dspark: Confidence-scheduled speculative decoding with semi-autoregressive generation, 2026. URL https://arxiv.org/abs/2607.05147.

Pierre Del Moral, Arnaud Doucet, and Ajay Jasra. Sequential monte carlo samplers. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 68(3):411–436, 2006.

Arnaud Doucet, Nando De Freitas, Neil James Gordon, et al. *Sequential Monte Carlo methods in practice*, volume 1. Springer, 2001.

Yann Dubois, Balázs Galambosi, Percy Liang, and Tatsunori B. Hashimoto. Length-controlled alpacaeval: A simple way to debias automatic evaluators, 2025. URL https://arxiv.org/abs/2404.04475.

Yongchang Hao and Lili Mou. Cactus: Constrained acceptance speculative sampling. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=lpUIkCAy9p.

Maximilian Holsman, Yukun Huang, and Bhuwan Dhingra. Fuzzy speculative decoding for a tunable accuracy-runtime tradeoff, 2025. URL https://arxiv.org/abs/2502.20704.

Kaixuan Huang, Xudong Guo, and Mengdi Wang. Specdec++: Boosting speculative decoding via adaptive candidate lengths, 2025. URL https://arxiv.org/abs/2405.19715.

Jonathan Hunter Huggins. *An information-theoretic analysis of resampling in sequential Monte Carlo*. PhD thesis, Massachusetts Institute of Technology, 2014. URL https://dspace.mit.edu/bitstream/handle/1721.1/91033/892649604-MIT.pdf.

Aayush Karan and Yilun Du. Reasoning with sampling: Your base model is smarter than you think, 2025. URL https://arxiv.org/abs/2510.14901.

Tanishq Kumar, Tri Dao, and Avner May. Speculative speculative decoding, 2026. URL https://arxiv.org/abs/2603.03251.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention, 2023. URL https://arxiv.org/abs/2309.06180.

Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Scott Wen tau Yih, Daniel Fried, Sida Wang, and Tao Yu. Ds-1000: A natural and reliable benchmark for data science code generation, 2022. URL https://arxiv.org/abs/2211.11501.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding, 2023. URL https://arxiv.org/abs/2211.17192.

Alexander K. Lew, Tan Zhi-Xuan, Gabriel Grand, and Vikash Mansinghka. Sequential monte carlo steering of large language models using probabilistic programs. In *ICML 2023 Workshop: Sampling and Optimization in Discrete Space*, 2023. URL https://openreview.net/forum?id=Ul2K0qXxXy.

Jinze Li, Yixing Xu, Guanchen Li, Shuo Yang, Jinfeng Xu, Xuanwu Yin, Dong Li, Edith C. H. Ngai, and Emad Barsoum. Training-free loosely speculative decoding: Accepting semantically correct drafts beyond exact match. *arXiv preprint arXiv:2511.22972*, 2025a. URL https://arxiv.org/abs/2511.22972.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-2: Faster inference of language models with dynamic draft trees, 2024. URL https://arxiv.org/abs/2406.16858.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-3: Scaling up inference acceleration of large language models via training-time test, 2025b. URL https://arxiv.org/abs/2503.01840.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle: Speculative sampling requires rethinking feature uncertainty, 2025c. URL https://arxiv.org/abs/2401.15077.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step, 2023. URL https://arxiv.org/abs/2305.20050.

João Loula, Benjamin LeBrun, Li Du, Ben Lipkin, Clemente Pasti, Gabriel Grand, Tianyu Liu, Yahya Emara, Marjorie Freedman, Jason Eisner, Ryan Cotterell, Vikash Mansinghka, Alexander K. Lew, Tim Vieira, and Timothy J. O'Donnell. Syntactic and semantic control of large language models via sequential monte carlo. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=xoXn62FzD0.

Christian A. Naesseth, Fredrik Lindsten, and Thomas B. Schön. Elements of sequential monte carlo, 2024. URL https://arxiv.org/abs/1903.04797.

NVIDIA Corporation. NVIDIA A100 Tensor Core GPU Datasheet. Technical report, NVIDIA Corporation, 2020. URL https://resources.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/a100-80gb-datasheet-update-a4-nvidia-1485612-r12-web.pdf. Accessed March 2026.

NVIDIA Corporation. NVIDIA Blackwell Platform Datasheet. Technical report, NVIDIA Corporation, 2024a. URL https://resources.nvidia.com/en-us/data-center/technologies/blackwell-architecture/. Accessed March 2026.

NVIDIA Corporation. NVIDIA H100 Tensor Core GPU Datasheet. Technical report, NVIDIA Corporation, 2024b. URL https://resources.nvidia.com/en-us-tensor-core/nvidia-tensor-core-gpu-datasheet. Accessed March 2026.

NVIDIA Corporation. NVIDIA H200 Tensor Core GPU Datasheet. Technical report, NVIDIA Corporation, 2024c. URL https://resources.nvidia.com/en-us/data-center/h200/. Accessed March 2026.

Llama Team. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Samuel Williams, Andrew Waterman, and David Patterson. Roofline: An insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2009. doi: 10.1145/1498765.1498785.

Vicky Xefteri, Tim Vieira, Ryan Cotterell, and Afra Amini. Syntactic control of language models by posterior inference. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 25350–25365, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.1300. URL https://aclanthology.org/2025.findings-acl.1300/.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for Transformer-Based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pp. 521–538. USENIX Association, 2022.

Stephen Zhao, Rob Brekelmans, Alireza Makhzani, and Roger Grosse. Probabilistic Inference in Language Models via Twisted Sequential Monte Carlo. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. Sglang: Efficient execution of structured language model programs, 2024. URL https://arxiv.org/abs/2312.07104.

A Speculative Decoding Algorithm

We give the full pseudocode for the standard speculative decoding procedure of Leviathan et al. (2023); Chen et al. (2023), referenced in § 2.2. Given a draft length K and the draft–target pair (q, p), each iteration proceeds in two phases. In the **draft** phase, given the string generated so far $\boldsymbol{x}$, the draft model generates a draft continuation $\boldsymbol{d}$ of at most K tokens autoregressively from $\overrightarrow{q}$. In the **verification** phase, the target model scores all draft tokens in a single forward pass. Each draft token d_j is then tested sequentially using the standard rejection sampling criterion: it is accepted with probability $\alpha_j = \min\big(1, \overrightarrow{p}(d_j \mid \boldsymbol{x}\,\boldsymbol{d}_{<j}) / \overrightarrow{q}(d_j \mid \boldsymbol{x}\,\boldsymbol{d}_{<j})\big)$. On the first rejection, the token is resampled from the residual distribution $\max\big(0,\ \overrightarrow{p}(\cdot \mid \boldsymbol{x}\,\boldsymbol{d}_{<j}) - \overrightarrow{q}(\cdot \mid \boldsymbol{x}\,\boldsymbol{d}_{<j})\big)$ and the draft is truncated. If every token passes, a bonus token is drawn from $\overrightarrow{p}(\cdot \mid \boldsymbol{x}\,\boldsymbol{d})$.

Algorithm 2 Speculative Decoding (one round)

Require: Prefix $\boldsymbol{x}$, draft model q, target model p, draft length K
Ensure: Extended string $\boldsymbol{x}\,\boldsymbol{d}$

— Draft —
$\boldsymbol{d} \leftarrow \varepsilon$
for $t = 1, \ldots, K$ **do**
  $d_t \sim \overrightarrow{q}(\cdot \mid \boldsymbol{x}\,\boldsymbol{d})$
  $\boldsymbol{d} \leftarrow \boldsymbol{d}\,d_t$
end for
— Verification —
for $j = 1, \ldots, K$ **do**
  $u \sim \mathrm{Uniform}(0, 1)$
  if $u > \min\big(1,\ \overrightarrow{p}(d_j \mid \boldsymbol{x}\,\boldsymbol{d}_{<j}) \,/\, \overrightarrow{q}(d_j \mid \boldsymbol{x}\,\boldsymbol{d}_{<j})\big)$ **then** ▷ reject
    $d_j \sim \max\big(0,\ \overrightarrow{p}(\cdot \mid \boldsymbol{x}\,\boldsymbol{d}_{<j}) - \overrightarrow{q}(\cdot \mid \boldsymbol{x}\,\boldsymbol{d}_{<j})\big)/Z_j$ ▷ Z_j normalizes
    $\boldsymbol{d} \leftarrow \boldsymbol{d}_{<j}\,d_j$ ▷ truncate and replace
    break
  else if $j = K$ **then** ▷ all tokens accepted
    $d \sim \overrightarrow{p}(\cdot \mid \boldsymbol{x}\,\boldsymbol{d})$ ▷ bonus token
    $\boldsymbol{d} \leftarrow \boldsymbol{d}\,d$
  end if
end for
return $\boldsymbol{x}\,\boldsymbol{d}$

B PROOFS

B.1 PROOF OF THEOREM 3.1

We analyze the SMC-SD variant that performs one multinomial resampling step per round, working with the multi-round target and particle approximation introduced in § 3.2. We first describe the variant precisely, decompose the per-round error into three terms, bound each term with a lemma, and finally close the recursion.

The once-per-round resampling variant. We adopt the **full-prefix convention**: a state $\boldsymbol{x} \in \mathcal{X}_t^*$ records its entire history, so it has a unique parent $\mathrm{par}(\boldsymbol{x}) \in \mathcal{X}_{t-1}^*$. EOS is handled by a fixed-length absorbing/padded representation. The initial target p_{-1} is the deterministic prompt distribution, and $\widehat{p}_{-1}^N = p_{-1}$. At the start of round t, the previous weighted particle law is $\widehat{p}_{t-1}^N = \sum_{m=1}^N \overline{w}_{t-1}^{(m)} \delta_{\boldsymbol{x}_{t-1}^{(m)}}$ with $\overline{w}_{t-1}^{(m)} \geq 0$ and $\sum_m \overline{w}_{t-1}^{(m)} = 1$. Let $\mathcal{F}_{t-1} \stackrel{\text{def}}{=} \sigma\big(\boldsymbol{x}_{t-1}^{1:N}, \overline{w}_{t-1}^{1:N}\big)$. The variant performs one multinomial resampling step per round: the resampled ancestors are drawn conditionally i.i.d.,

$$\widetilde{\boldsymbol{x}}_t^{(1)}, \ldots, \widetilde{\boldsymbol{x}}_t^{(N)} \mid \mathcal{F}_{t-1} \overset{\text{i.i.d.}}{\sim} \widehat{p}_{t-1}^N, \tag{9}$$

and each is extended by one block drawn from the draft model, conditionally independently given the ancestors:

$$\boldsymbol{x}_t^{(n)} \mid \widetilde{\boldsymbol{x}}_t^{(n)}, \mathcal{F}_{t-1} \sim q(\cdot \mid \widetilde{\boldsymbol{x}}_t^{(n)}), \qquad n = 1, \ldots, N. \tag{10}$$

The corresponding block weight is $w_t^{(n)} \stackrel{\text{def}}{=} p(\boldsymbol{x}_t^{(n)} \mid \widetilde{\boldsymbol{x}}_t^{(n)}) / q(\boldsymbol{x}_t^{(n)} \mid \widetilde{\boldsymbol{x}}_t^{(n)})$, and we define the unnormalized weighted empirical measure and its total mass,

$$U_t^N \stackrel{\text{def}}{=} \frac{1}{N} \sum_{n=1}^N w_t^{(n)} \delta_{\boldsymbol{x}_t^{(n)}}, \qquad S_t^N \stackrel{\text{def}}{=} U_t^N(1) = \frac{1}{N} \sum_{n=1}^N w_t^{(n)}, \tag{11}$$

so that the particle law after round t is $\widehat{p}_t^N = \sum_n \overline{w}_t^{(n)} \delta_{\boldsymbol{x}_t^{(n)}} = U_t^N / S_t^N$, with $\overline{w}_t^{(n)}$ as in (5). We also write $\mu_t^N \stackrel{\text{def}}{=} \frac{1}{N} \sum_{n=1}^N p(\cdot \mid \widetilde{\boldsymbol{x}}_t^{(n)})$ for the random mixture of target conditionals over the resampled ancestors. Finally, for a (signed) measure ν on $\mathcal{X}_{t-1}^*$, let $\mathcal{E}_t(\nu)$ denote the measure on $\mathcal{X}_t^*$ obtained by appending one block under the target conditional,

$$\mathcal{E}_t(\nu)(\boldsymbol{x}) \stackrel{\text{def}}{=} \nu(\mathrm{par}(\boldsymbol{x}))\, p(\boldsymbol{x} \mid \mathrm{par}(\boldsymbol{x})), \qquad \boldsymbol{x} \in \mathcal{X}_t^*, \tag{12}$$

so that the target recursion reads $p_t = \mathcal{E}_t(p_{t-1})$, and, since $\widetilde{\boldsymbol{x}}_t^{(n)} \sim \widehat{p}_{t-1}^N$, $\mathbb{E}[\mu_t^N \mid \mathcal{F}_{t-1}] = \mathcal{E}_t(\widehat{p}_{t-1}^N)$.

Three-term decomposition. Since $p_t = \mathcal{E}_t(p_{t-1})$, the per-round error decomposes as

$$\widehat{p}_t^N - p_t = \underbrace{\big(\widehat{p}_t^N - \mu_t^N\big)}_{\text{(i) importance error}} + \underbrace{\big(\mu_t^N - \mathcal{E}_t(\widehat{p}_{t-1}^N)\big)}_{\text{(ii) resampling variance}} + \underbrace{\big(\mathcal{E}_t(\widehat{p}_{t-1}^N) - \mathcal{E}_t(p_{t-1})\big)}_{\text{(iii) propagated error}}, \tag{13}$$

namely the importance error around the random mixture μ_t^N, the variance introduced by multinomial resampling of ancestors, and the previous-round error propagated through one block extension. The next three lemmas bound these terms.

Lemma B.1 (Mixed-prefix importance step)**.** *Let* $\mathcal{G}_t \stackrel{\text{def}}{=} \sigma\big(\mathcal{F}_{t-1}, \widetilde{\boldsymbol{x}}_t^{1:N}\big)$. *Then*

$$\mathbb{E}\Big[\big\|\widehat{p}_t^N - \mu_t^N\big\|_2^2 \,\Big|\, \mathcal{G}_t\Big] \leq \frac{4}{N^2} \sum_{n=1}^N \big(1 + \chi_t^2(\widetilde{\boldsymbol{x}}_t^{(n)})\big), \tag{14}$$

and consequently

$$\mathbb{E}\Big[\big\|\widehat{p}_t^N - \mu_t^N\big\|_2^2 \,\Big|\, \mathcal{F}_{t-1}\Big] \leq \frac{4}{N}\, \widehat{p}_{t-1}^N(1 + \chi_t^2) \leq \frac{4(1 + \bar{\chi}_t^2)}{N}. \tag{15}$$

Proof. For any $\boldsymbol{x} \in \mathcal{X}_t^*$, $U_t^N(\boldsymbol{x}) = \frac{1}{N} \sum_n w_t^{(n)} \mathbb{1}\{\boldsymbol{x}_t^{(n)} = \boldsymbol{x}\}$. Conditionally on $\mathcal{G}_t$, the summands indexed by n are independent, and since $\boldsymbol{x}_t^{(n)} \sim q(\cdot \mid \widetilde{\boldsymbol{x}}_t^{(n)})$ and $w_t^{(n)} = p(\boldsymbol{x}_t^{(n)} \mid \widetilde{\boldsymbol{x}}_t^{(n)}) / q(\boldsymbol{x}_t^{(n)} \mid \widetilde{\boldsymbol{x}}_t^{(n)})$,

$$\mathbb{E}\big[w_t^{(n)} \mathbb{1}\{\boldsymbol{x}_t^{(n)} = \boldsymbol{x}\} \mid \mathcal{G}_t\big] = \sum_{\boldsymbol{x}'' \in \mathcal{X}_t^*} q(\boldsymbol{x}'' \mid \widetilde{\boldsymbol{x}}_t^{(n)}) \frac{p(\boldsymbol{x}'' \mid \widetilde{\boldsymbol{x}}_t^{(n)})}{q(\boldsymbol{x}'' \mid \widetilde{\boldsymbol{x}}_t^{(n)})} \mathbb{1}\{\boldsymbol{x}'' = \boldsymbol{x}\} = p(\boldsymbol{x} \mid \widetilde{\boldsymbol{x}}_t^{(n)}), \tag{16}$$

so $\mathbb{E}[U_t^N \mid \mathcal{G}_t] = \mu_t^N$. The centered vector summands $w_t^{(n)}\delta_{\boldsymbol{x}_t^{(n)}} - p(\cdot \mid \widetilde{\boldsymbol{x}}_t^{(n)})$ are conditionally independent with conditional mean zero, so the cross terms vanish in the squared norm:

$$\mathbb{E}\Big[\big\|U_t^N - \mu_t^N\big\|_2^2 \,\Big|\, \mathcal{G}_t\Big] = \frac{1}{N^2}\sum_{n=1}^{N}\mathbb{E}\Big[\big\|w_t^{(n)}\delta_{\boldsymbol{x}_t^{(n)}} - p(\cdot \mid \widetilde{\boldsymbol{x}}_t^{(n)})\big\|_2^2 \,\Big|\, \mathcal{G}_t\Big] \le \frac{1}{N^2}\sum_{n=1}^{N}\mathbb{E}\Big[\big(w_t^{(n)}\big)^2 \,\Big|\, \mathcal{G}_t\Big]. \tag{17}$$

By (6), $\mathbb{E}[(w_t^{(n)})^2 \mid \mathcal{G}_t] = \mathbb{E}_{\boldsymbol{x}\sim q(\cdot\mid\widetilde{\boldsymbol{x}}_t^{(n)})}[(p(\boldsymbol{x} \mid \widetilde{\boldsymbol{x}}_t^{(n)})/q(\boldsymbol{x} \mid \widetilde{\boldsymbol{x}}_t^{(n)}))^2] = 1 + \chi_t^2(\widetilde{\boldsymbol{x}}_t^{(n)})$, hence

$$\mathbb{E}\Big[\big\|U_t^N - \mu_t^N\big\|_2^2 \,\Big|\, \mathcal{G}_t\Big] \le \frac{1}{N^2}\sum_{n=1}^{N}\big(1 + \chi_t^2(\widetilde{\boldsymbol{x}}_t^{(n)})\big). \tag{18}$$

Next, the weights are proper importance weights: $\mathbb{E}[w_t^{(n)} \mid \mathcal{G}_t] = \sum_{\boldsymbol{x}} p(\boldsymbol{x} \mid \widetilde{\boldsymbol{x}}_t^{(n)}) = 1$. Since the $w_t^{(n)}$ are conditionally independent,

$$\mathbb{E}\big[(S_t^N - 1)^2 \mid \mathcal{G}_t\big] = \frac{1}{N^2}\sum_{n=1}^{N}\mathbb{E}\big[(w_t^{(n)} - 1)^2 \mid \mathcal{G}_t\big] = \frac{1}{N^2}\sum_{n=1}^{N}\chi_t^2(\widetilde{\boldsymbol{x}}_t^{(n)}) \le \frac{1}{N^2}\sum_{n=1}^{N}\big(1 + \chi_t^2(\widetilde{\boldsymbol{x}}_t^{(n)})\big). \tag{19}$$

Because $U_t^N = S_t^N\,\widehat{p}_t^N$, we have the normalization identity $\widehat{p}_t^N - \mu_t^N = (U_t^N - \mu_t^N) - (S_t^N - 1)\widehat{p}_t^N$. As $\widehat{p}_t^N$ is a probability vector, $\|\widehat{p}_t^N\|_2^2 = \sum_{\boldsymbol{x}} \widehat{p}_t^N(\boldsymbol{x})^2 \le \sum_{\boldsymbol{x}} \widehat{p}_t^N(\boldsymbol{x}) = 1$, so

$$\big\|\widehat{p}_t^N - \mu_t^N\big\|_2^2 \le 2\big\|U_t^N - \mu_t^N\big\|_2^2 + 2(S_t^N - 1)^2. \tag{20}$$

Combining (18) and (19) proves (14). Finally, by (9) the ancestors $\widetilde{\boldsymbol{x}}_t^{1:N}$ are conditionally i.i.d. from $\widehat{p}_{t-1}^N$, so taking conditional expectation given $\mathcal{F}_{t-1}$,

$$\mathbb{E}\Big[\frac{4}{N^2}\sum_{n=1}^{N}\big(1 + \chi_t^2(\widetilde{\boldsymbol{x}}_t^{(n)})\big) \,\Big|\, \mathcal{F}_{t-1}\Big] = \frac{4}{N}\,\widehat{p}_{t-1}^N(1 + \chi_t^2) \le \frac{4(1 + \bar{\chi}_t^2)}{N}, \tag{21}$$

which gives (15). ■

Lemma B.2 (Ancestor-resampling variance)**.** *Conditionally on $\mathcal{F}_{t-1}$,*

$$\mathbb{E}\Big[\big\|\mu_t^N - \mathcal{E}_t(\widehat{p}_{t-1}^N)\big\|_2^2 \,\Big|\, \mathcal{F}_{t-1}\Big] \le \frac{1}{N}. \tag{22}$$

Proof. Conditionally on $\mathcal{F}_{t-1}$, the vectors $p(\cdot \mid \widetilde{\boldsymbol{x}}_t^{(n)})$ are i.i.d. $\ell^2(\mathcal{X}_t^*)$-valued with mean $\mathbb{E}[p(\cdot \mid \widetilde{\boldsymbol{x}}_t^{(n)}) \mid \mathcal{F}_{t-1}] = \mathcal{E}_t(\widehat{p}_{t-1}^N)$, and $\mu_t^N = \frac{1}{N}\sum_n p(\cdot \mid \widetilde{\boldsymbol{x}}_t^{(n)})$ is their average. By the variance identity for i.i.d. averages (Naesseth et al., 2024),

$$\begin{aligned}&\mathbb{E}\Big[\big\|\mu_t^N - \mathcal{E}_t(\widehat{p}_{t-1}^N)\big\|_2^2 \,\Big|\, \mathcal{F}_{t-1}\Big] \\ &= \frac{1}{N}\,\mathbb{E}\Big[\big\|p(\cdot \mid \widetilde{\boldsymbol{x}}_t^{(1)}) - \mathcal{E}_t(\widehat{p}_{t-1}^N)\big\|_2^2 \,\Big|\, \mathcal{F}_{t-1}\Big] \\ &\le \frac{1}{N}\,\mathbb{E}\Big[\big\|p(\cdot \mid \widetilde{\boldsymbol{x}}_t^{(1)})\big\|_2^2 \,\Big|\, \mathcal{F}_{t-1}\Big].\end{aligned} \tag{23}$$

For every $\boldsymbol{x}'$, $p(\cdot \mid \boldsymbol{x}')$ is a probability distribution, so $\|p(\cdot \mid \boldsymbol{x}')\|_2^2 = \sum_{\boldsymbol{x}} p(\boldsymbol{x} \mid \boldsymbol{x}')^2 \le \sum_{\boldsymbol{x}} p(\boldsymbol{x} \mid \boldsymbol{x}') = 1$, and the right-hand side is at most $1/N$. ■

Lemma B.3 (Append-only contraction)**.** *Under the full-prefix convention, every $\boldsymbol{x} \in \mathcal{X}_t^*$ has a unique parent $\mathrm{par}(\boldsymbol{x}) \in \mathcal{X}_{t-1}^*$. Then for any signed measure Δ on $\mathcal{X}_{t-1}^*$,*

$$\|\mathcal{E}_t(\Delta)\|_2^2 \le \|\Delta\|_2^2. \tag{24}$$

Proof. By the definition (12), $\mathcal{E}_t(\Delta)(\boldsymbol{x}) = \Delta(\mathrm{par}(\boldsymbol{x}))\, p(\boldsymbol{x} \mid \mathrm{par}(\boldsymbol{x}))$, since no parent other than $\mathrm{par}(\boldsymbol{x})$ can produce $\boldsymbol{x}$. Hence

$$\|\mathcal{E}_t(\Delta)\|_2^2 = \sum_{\boldsymbol{x}\in\mathcal{X}_t^*} \Delta(\mathrm{par}(\boldsymbol{x}))^2\, p(\boldsymbol{x} \mid \mathrm{par}(\boldsymbol{x}))^2 = \sum_{\boldsymbol{x}'\in\mathcal{X}_{t-1}^*} \Delta(\boldsymbol{x}')^2 \sum_{\boldsymbol{x}:\mathrm{par}(\boldsymbol{x})=\boldsymbol{x}'} p(\boldsymbol{x} \mid \boldsymbol{x}')^2. \tag{25}$$

For each $\boldsymbol{x}'$, $\sum_{\boldsymbol{x}:\mathrm{par}(\boldsymbol{x})=\boldsymbol{x}'} p(\boldsymbol{x} \mid \boldsymbol{x}')^2 \le \sum_{\boldsymbol{x}:\mathrm{par}(\boldsymbol{x})=\boldsymbol{x}'} p(\boldsymbol{x} \mid \boldsymbol{x}') = 1$, so $\|\mathcal{E}_t(\Delta)\|_2^2 \le \sum_{\boldsymbol{x}'} \Delta(\boldsymbol{x}')^2 = \|\Delta\|_2^2$. ■

We now restate and prove the theorem.

Theorem 3.1 (SMC-SD MSE bound)**.** *Consider the SMC-SD variant that performs one multinomial resampling step at the start of each round (§ B). If the per-round divergences are uniformly bounded, $\bar{\chi}_t^2 \le \bar{\chi}^2$ for all t, then for a fixed generation horizon T,*

$$\mathbb{E}\big[\|\widehat{p}_T^N - p_T\|_2^2\big] \lesssim_T \frac{1+\bar{\chi}^2}{N}. \tag{7}$$

Proof. Throughout the proof we write $e_t \overset{\text{def}}{=} \mathbb{E}\big[\|\widehat{p}_t^N - p_t\|_2^2\big]$ as a shorthand, with $e_{-1} = 0$ under the convention $\widehat{p}_{-1}^N = p_{-1}$. Starting from the decomposition (13) and using $\|a+b+c\|_2^2 \le 3\|a\|_2^2 + 3\|b\|_2^2 + 3\|c\|_2^2$,

$$e_t \le 3\,\mathbb{E}\big\|\widehat{p}_t^N - \mu_t^N\big\|_2^2 + 3\,\mathbb{E}\big\|\mu_t^N - \mathcal{E}_t(\widehat{p}_{t-1}^N)\big\|_2^2 + 3\,\mathbb{E}\big\|\mathcal{E}_t(\widehat{p}_{t-1}^N) - \mathcal{E}_t(p_{t-1})\big\|_2^2. \tag{26}$$

Lemmas B.1 and B.2 bound the first two terms by $4(1+\bar{\chi}_t^2)/N$ and $1/N$ in expectation, and Lemma B.3 applied with $\Delta = \widehat{p}_{t-1}^N - p_{t-1}$ (using linearity of $\mathcal{E}_t$) bounds the third by e_{t-1}. Hence, since $1 \le 1+\bar{\chi}_t^2$,

$$e_t \le 3\cdot\frac{4(1+\bar{\chi}_t^2)}{N} + 3\cdot\frac{1}{N} + 3\,e_{t-1} \le 3\,e_{t-1} + \frac{15(1+\bar{\chi}_t^2)}{N}. \tag{27}$$

Iterating (27) from $e_{-1} = 0$ gives, for every horizon T,

$$e_T \le \frac{15}{N}\sum_{t=0}^{T} 3^{\,T-t}\,(1+\bar{\chi}_t^2). \tag{28}$$

If $\bar{\chi}_t^2 \le \bar{\chi}^2$ uniformly in t, then $\sum_{t=0}^{T} 3^{\,T-t}(1+\bar{\chi}_t^2) \le (1+\bar{\chi}^2)\sum_{t=0}^{T} 3^{\,T-t} \le \frac{3}{2}\,3^T\,(1+\bar{\chi}^2)$, so for a fixed horizon T,

$$\mathbb{E}\big[\|\widehat{p}_T^N - p_T\|_2^2\big] = e_T \lesssim_T \frac{1+\bar{\chi}^2}{N}, \tag{29}$$

which is (7). ■

C Speed-up Summary Table

Table 1: SMC-SD speed-up over standard SD at two operating points. *Iso-accuracy*: fastest SMC-SD config within 3pp of SD accuracy. *Max speed*: fastest config within 10pp. The Llama configuration uses a 1B draft model and 8B target model for both SMC-SD and standard SD. The Qwen configuration uses a 0.5B draft model and 14B target for standard SD and a 3B draft and 14B target for SMC-SD.

		SD		SMC-SD (iso-accuracy)			SMC-SD (max speed)		
Benchmark	Models	Acc	TPS	Acc	TPS	Speed	Acc	TPS	Speed
GSM8K	Llama	78	250	81	290	1.16×	72	380	1.52×
	Qwen	90	170	88.5	241	1.42×	88.5	241	1.42×
MATH500	Llama	52	250	53	380	1.52×	53	380	1.52×
	Qwen	75	200	74	270	1.35×	74	270	1.35×
AlpacaEval	Llama	89	190	88	270	1.42×	84	350	1.84×
	Qwen	92	112	90	210	1.88×	88	250	2.23×
DS1000	Llama	26.5	150	26.5	379.3	2.53×	23.2	502.9	3.35×
	Qwen	35	119	33	244	2.05×	31	300	2.52×

D BATCH-SIZE SCALING

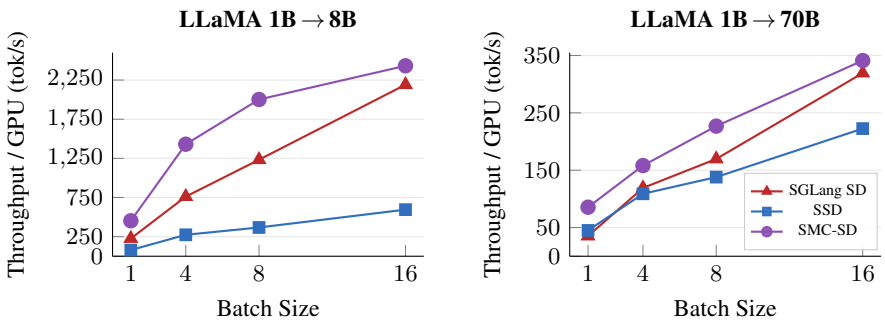


Figure 5: Throughput per GPU vs. batch size on ShareGPT for SGLang SD, SSD, and SMC-SD. 1B→8B uses 1 GPU for SGLang SD and SMC-SD, 2 GPUs for SSD. 1B→70B uses 4 H100 GPUs for SGLang SD and SMC-SD, 5 H100 GPUs for SSD. SMC-SD achieves the highest per-GPU throughput across most batch sizes for both model pairs.

E LOSSY TREE SPECULATIVE DECODING: FULL RESULTS

All throughputs reported here are decode-only: a calibration pass over the identical prompts with a one-token generation budget measures per-request prefill time, which is subtracted from the benchmark wall time. At temperature 0.7 the standalone Llama 3.2-1B draft is a substantially stronger tree proposer than the public EAGLE3 heads (mean accepted length 4.5–5.5 versus 1.9–3.0), and relaxing the acceptance threshold buys tree SD up to 23% additional throughput on the 8B target. Cactus raises the mean accepted length to 8.7 (8B) and 6.4 (70B) tokens per round, giving 1.74× and 1.56× the throughput of vLLM's lossless speculative sampling.

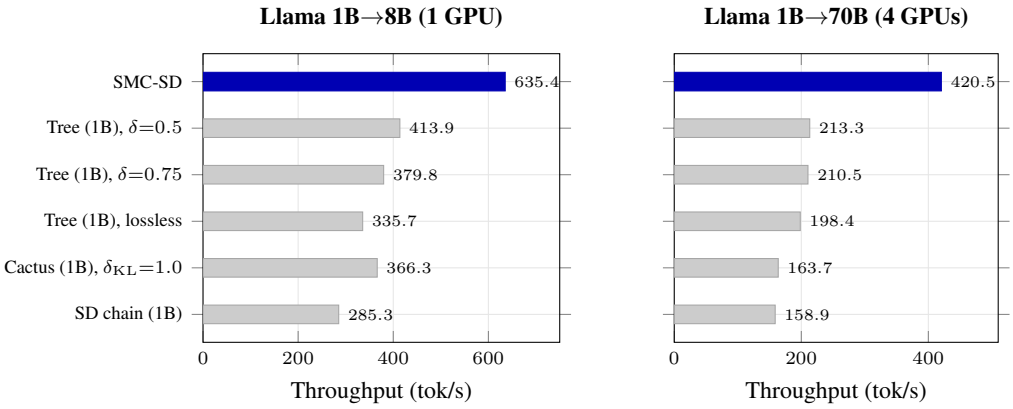


Figure 6: SMC-SD versus lossy tree speculative decoding on ShareGPT at batch size 1 (temperature 0.7, NVIDIA B200 GPUs; 70B target uses TP=4). Throughput is decode-only: per-request prefill time, measured by a one-token calibration pass over the identical prompts, is excluded. "Tree" rows use SGLang's EAGLE-style top-k=10 token tree (6 draft steps, 32 draft tokens per verification) with relaxed acceptance threshold δ (lower is lossier; "lossless" is exact tree verification), proposed by a standalone Llama 3.2-1B draft model. "SD chain" is SGLang's standard (lossless) chain speculative decoding with the 1B draft. "Cactus" (Hao & Mou, 2026) runs in its native vLLM implementation (chain, k=10, KL budget δ_{KL}=1.0) on the identical request set. Targets are Llama 3.1-8B (left) and Llama 3.3-70B (right). SMC-SD leads the best lossy tree configuration by 1.54× and 1.97×, respectively. Full numbers, including a trained EAGLE3 draft head as the tree proposer and Cactus's in-engine lossless baseline, are in Table 2 (§ E).

Table 2: End-to-end and decode-only output throughput, and mean accepted length per verification round, for the lossy tree speculative decoding comparison of Figure 6 (ShareGPT, batch size 1, temperature 0.7, NVIDIA B200 GPUs; the 70B target uses TP=4). End-to-end throughput divides generated tokens by total wall time including prefill; decode-only excludes per-request prefill time, measured by a calibration pass over the identical prompts with a one-token generation budget. "Tree" rows use SGLang's EAGLE-style top-k=10 token tree (6 draft steps, 32 draft tokens per verification) with relaxed acceptance threshold δ; δ=1 ("lossless") is exact tree verification. The EAGLE3 rows use the public trained EAGLE3 draft heads for each target. The Cactus rows (Hao & Mou, 2026) run in Cactus's native vLLM implementation on the identical request set, with chain drafting (k=10 drafted tokens per round). Engine differences mean vLLM and SGLang rows are not strictly comparable in absolute terms. SMC-SD has no acceptance step: all $K+1$ tokens advance each round via importance-weighted resampling.

	1B→8B (1 GPU)			1B→70B (4 GPUs)		
Method	e2e	decode	Accept	e2e	decode	Accept
SD chain (1B draft), lossless	277.8	285.3	2.86	153.5	158.9	2.88
Tree (EAGLE3 head), δ=0.5	377.7	384.1	2.98	99.4	100.5	1.90
Tree (1B draft), lossless	329.5	335.7	4.47	192.8	198.4	4.63
Tree (1B draft), δ=0.75	372.0	379.8	5.05	204.0	210.5	4.92
Tree (1B draft), δ=0.5	404.3	413.9	5.48	206.4	213.3	4.99
Cactus (1B draft), δ_{KL}=1.0	360.1	366.3	8.74	160.7	163.7	6.43
SMC-SD	**591.6**	**635.4**	—	**370.3**	**420.5**	—

Table 3: Per-request prefill time and peak target-model KV-cache occupancy for the methods of Table 2. Prefill is measured on the ShareGPT batch-size-1 workload. KV occupancy is the peak number of occupied token slots in the target model's KV pool for a single request with a 256-token prompt and 512-token generation (Llama 3.1-8B), sampled at decode-step boundaries for SMC-SD and computed exactly for the chain/tree baselines (prompt + generation + in-flight draft tokens). SMC-SD's occupancy is 1.45× chain SD—far below the naive N=6×—because the reference-counted allocator shares each particle's common ancestral history and resampling repeatedly coalesces particles.

	Prefill (ms/request)		Peak KV occupancy
Method	8B	70B	(token slots, 8B, 256→512)
SD chain (1B draft)	25	57	773
Tree (1B draft), δ=0.5	16	41	800
Tree (EAGLE3 head), δ=0.5	13	37	800
Cactus (1B draft)	14	33	778
SMC-SD	22	67	1123

F Comparison against Modern Speculative Decoders

Table 4 compares SMC-SD against recent speculative decoders on the *same* target model, Qwen3-14B. We only use Qwen3-14B here, and not Llama models, because the public DSpark and EAGLE3 draft checkpoints are trained specifically for it.

Table 4: Output throughput (tokens/s) on ShareGPT, Qwen3-14B target, one NVIDIA H100-80GB.

	Throughput (tok/s) at batch size B			
Method	1	4	8	16
Autoregressive (no speculation)	90.6	334.5	606.6	1022.7
Speculative decoding (0.6B draft)	131.7	473.3	779.7	960.3
EAGLE3	113.0	417.0	726.2	1202.5
DSpark (block 7)	195.4	697.3	1167.2	**1981.2**
SMC-SD	**380.5**	**1078.8**	**1422.8**	1641.7

G Hardware Utilization and Speedup Proofs

Let $\mathcal{P}$ denote the target model's parameter count, b the bytes per parameter, $\rho = \mathcal{P}_q/\mathcal{P}_p$ the draft-to-target parameter ratio, $c = T_q/T_p \ll 1$ the draft-to-target decoding-time ratio. Let B denote the number of independent user requests served concurrently (the *batch size*), and let BN denote the *effective batch size*—the total number of particles across all requests. In the memory-bound regime $c \approx \rho$. In one SMC-SD step, the draft model runs K sequential forward passes (one per token position, with all BN particles batched), taking cT_pK seconds. The target model scores all $BN(K+1)$ tokens (including bonus tokens) in a single forward pass. The resampling step is negligible.

G.1 Arithmetic Intensity

Lemma G.1 (Target-pass arithmetic intensity)**.** *The arithmetic intensity of the target model's forward pass in one SMC-SD cycle is*

$$AI_{target} = \frac{2BN(K+1)}{b} \ \textit{FLOPs/byte}. \tag{30}$$

Proof. Using the standard approximation that a forward pass through a model with $\mathcal{P}$ parameters requires $2\mathcal{P}$ FLOPs (one multiply-add per parameter), the target pass processes $BN(K+1)$ tokens in one invocation—BNK draft tokens plus BN bonus tokens carried over from the previous round—for a total of $2\mathcal{P} \cdot BN(K+1)$ FLOPs. The bytes transferred are $b\mathcal{P}$ (loading the weights once), giving $\text{AI}_{\text{target}} = 2\mathcal{P} \cdot BN(K+1)/(b\mathcal{P}) = 2BN(K+1)/b$. For FP16 ($b$=2) this reduces to $BN(K+1)$. For comparison, autoregressive decoding achieves $\text{AI} = 2B/b$ and standard SD achieves $\text{AI} = 2B(K+1)/b$ when all K draft tokens are accepted. When $\text{AI}_{\text{target}}$ exceeds the hardware's ridge point (peak FLOPs / peak bandwidth), the target pass becomes compute-bound and further increasing the effective batch size BN no longer comes for free. ■

Lemma G.2 (Full-cycle arithmetic intensity)**.** *Over a full SMC-SD cycle (including K draft forward passes), the arithmetic intensity is*

$$AI_{cycle} = \frac{2BN(\rho K+K+1)}{b(\rho K+1)} \approx \frac{2BN(K+1)}{b} \quad \textit{when } \rho \ll 1. \tag{31}$$

Proof. The total FLOPs over one cycle are $2\mathcal{P}_q \cdot BNK + 2\mathcal{P} \cdot BN(K+1) = 2\mathcal{P}\, BN(\rho K + K + 1)$: the draft model performs K forward passes each processing BN tokens, and the target model performs one pass processing $BN(K+1)$ tokens. The total bytes transferred are $b\mathcal{P}_q \cdot K + b\mathcal{P} = b\mathcal{P}(\rho K + 1)$: the draft weights are loaded K times and the target weights once. Dividing gives $\text{AI}_{\text{cycle}} = 2BN(\rho K + K + 1)/(b(\rho K + 1))$. When $\rho \ll 1$, the draft-model terms become negligible and this simplifies to $2BN(K+1)/b$. ■

G.2 Speedup Proofs

Lemma G.3 (Memory-bound speed-up)**.** *If the target forward pass is memory bandwidth-bound (i.e. $BN(K+1) \le R$, where R is the hardware ridge-point batch size), the SMC-SD speed-up over*

autoregressive decoding is

$$S^{(mem)} = \frac{K+1}{cK+1}, \tag{32}$$

independent of the effective batch size BN.

Proof. When the target forward pass is memory bandwidth-bound, loading the model weights dominates and the verification cost is approximately constant regardless of BN. The cycle time is

$$t^{(\text{mem})}_{\text{SMC-SD step}} = T_p\big(cK+1\big). \tag{33}$$

Since autoregressive decoding takes T_pK seconds for K tokens and each SMC-SD cycle produces $K{+}1$ tokens per request, the speed-up is $S^{(\text{mem})} = (K{+}1)/(cK+1)$. Adding more particles or requests is "free" because the target weights are loaded only once regardless of BN. ■

Lemma G.4 (Compute-bound speed-up)**.** *If the target forward pass is compute-bound (i.e. $BN(K{+}1) > R$), the SMC-SD speed-up over autoregressive decoding is*

$$S^{(comp)} = \frac{K+1}{\hat{c}(BN)\,K + BN(K{+}1)/R}, \tag{34}$$

where $\hat{c}(BN) = \max(c,\, c\cdot BN/R)$ is the effective draft cost ratio and R is the hardware ridge-point batch size in tokens.

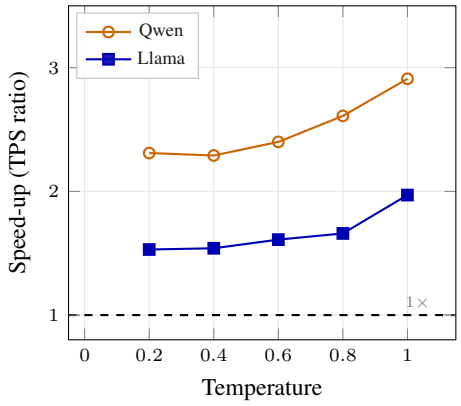
Proof. As $BN(K{+}1)$ grows, the arithmetic intensity of the target pass eventually exceeds the hardware ridge point, and the forward pass becomes compute-bound. In this regime, the target verification time scales linearly with the number of tokens processed. Let $T_p^{(1)}$ denote the (memory-bound) time to decode a single token with the target model. The compute-bound verification of $BN(K{+}1)$ tokens takes approximately $T_p^{(1)} \cdot BN(K{+}1)/R$ seconds, where $R = \text{peak FLOPs} \,/\, \text{peak bandwidth} \cdot b/2$ tokens (with b bytes per parameter). The draft model, which processes BN tokens in parallel at each of its K steps, may itself become compute-bound if $BN > R$. In the compute-bound regime, the draft model's per-step cost scales as $c \cdot BN/R$ relative to $T_p^{(1)}$, since the draft model has $c \approx \rho$ times the cost of the target per token. Writing $\hat{c}(BN) = \max(c,\, c\cdot BN/R)$ for the effective draft cost ratio, the cycle time becomes

$$t^{(\text{comp})}_{\text{SMC-SD step}} = T_p^{(1)}\left(\hat{c}(BN)\,K + \frac{BN(K{+}1)}{R}\right). \tag{35}$$

Since autoregressive decoding takes $T_p^{(1)}$ per token, the speed-up is $S^{(\text{comp})} = (K{+}1)/(\hat{c}(BN)K + BN(K{+}1)/R)$. ■

H Speed-up Under Declining Acceptance Rate

Standard SD's throughput is tightly coupled to the draft–target acceptance rate, which drops as sampling temperature increases (or more generally, as the draft and target distributions diverge). SMC-SD, by contrast, always outputs $K{+}1$ tokens per round regardless of acceptance, so its throughput is largely temperature-invariant. Figure 7 plots the SMC-SD / SD throughput ratio across temperatures: the speed-up grows from $\sim$1.5$\times$ at $T{=}0.2$ to $\sim$3$\times$ at $T{=}1.0$. On average across temperatures, SMC-SD sequences incur only a $\sim$5% increase in negative log-likelihood under the target model compared to exact SD, confirming that the approximation quality remains high even as the throughput advantage widens.



	Qwen 0.5B→14B			Llama 1B→8B		
T	SD	SMC	×	SD	SMC	×
0.2	133	307	2.3	222	340	1.5
0.4	130	298	2.3	218	335	1.5
0.6	125	300	2.4	202	325	1.6
0.8	115	300	2.6	190	315	1.7
1.0	103	300	2.9	158	312	2.0

Figure 7: SMC-SD / SD throughput ratio vs. temperature (left) and raw throughput in tokens/s (right). SMC-SD maintains near-constant TPS while SD degrades as acceptance rates fall, widening the speed-up from $\sim$1.5$\times$ at T=0.2 to $\sim$3$\times$ at T=1.0. On average, SMC-SD sequences incur only a $\sim$5% increase in NLL under the target model.

I Power Sampling

One advantage of SMC is that it allows us to sample approximately from un-normalized target distributions whose normalizing constants are intractable. One such target distribution is the *power distribution* (Karan & Du, 2025), given by $p_{\text{pow}}(\boldsymbol{x}) \propto p(\boldsymbol{x})^{\alpha}$ where $\boldsymbol{x} \in \mathcal{X}^*$. In contrast to temperature sampling, which tempers the distribution of each *token*, power sampling tempers the distribution over entire *strings*. Applying SMC-SD to sample approximately from this distribution is simply a matter of weighting the draft model q's generated tokens by

$$w_t^{(n)} = \frac{\overrightarrow{p}(x_t^{(n)} \mid \boldsymbol{x}_{<t}^{(n)})^{\alpha}}{\overrightarrow{q}(x_t^{(n)} \mid \boldsymbol{x}_{<t}^{(n)})}. \tag{36}$$

Prior work has proposed using SMC to sample approximately from the power distribution. PowerSMC (Azizi et al., 2026) proposes generating particles via temperature sampling

$$x_t^{(n)} \sim \frac{\overrightarrow{p}(\cdot \mid \boldsymbol{x}_{<t}^{(n)})^{\alpha}}{\sum_{x \in \mathcal{X}} \overrightarrow{p}(x \mid \boldsymbol{x}_{<t}^{(n)})^{\alpha}}, \tag{37}$$

weighting with importance weight equal to

$$w_t^{(n)} = \sum_{x \in \mathcal{X}} \overrightarrow{p}(x \mid \boldsymbol{x}_{<t}^{(n)})^{\alpha}, \tag{38}$$

and resampling after each batch of tokens is generated (if the ESS is less than a cutoff). The PowerSMC procedure can be expressed as a special case of the SMC-SD framework where $K = 1$, the bonus token is removed, and the target and draft models are the same. The authors prove that temperature sampling from the target model provides the locally variance-minimizing proposal for sampling from the power distribution. Figure 8 shows that sampling from the power distribution of an 8B Llama model with a 1B Llama draft model yields a 10% higher Pass@1 accuracy over the baseline target model. Accuracy for Pass@k for $k > 1$ is greater than or equal to the baseline target model as well.

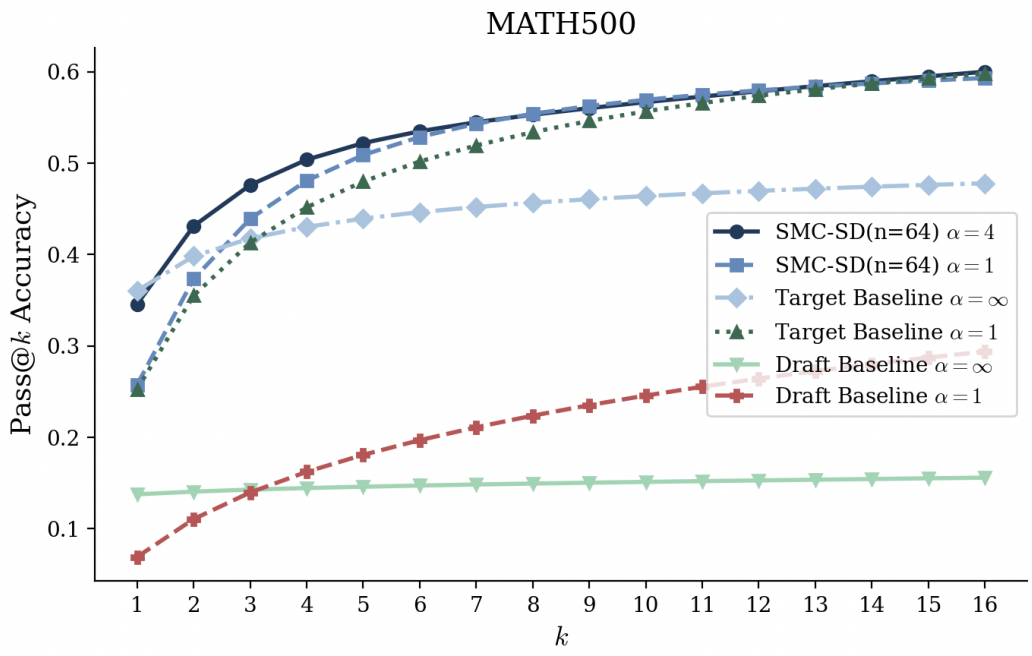


Figure 8: Power Sampling helps improve Pass@k quality

J More Qualitative Examples

GSM8K.

Llama 1B→8B

Question: Claire makes a 3 egg omelet every morning for breakfast. How many dozens of eggs will she eat in 4 weeks?

SMC-SD: Step 1: Convert 4 weeks to days — 4 weeks × 7 days/week = 28 days. Step 2: Calculate the number of eggs in 28 days — 28 days × 3 eggs/day = 84 eggs. Step 3: Convert the number of eggs from eggs to dozens — 84 eggs / 12 eggs/dozen = 7 dozens. The final answer is: 7.

Qwen 3B→14B

Question: A new program had 60 downloads in the first month. The number of downloads in the second month was three times as many as the downloads in the first month, but then reduced by 30% in the third month. How many downloads did the program have total over the three months?

SMC-SD: First month: 60 downloads. Second month: $3 \times 60 = 180$ downloads. Third month: reduced by 30%, so $180 \times 0.70 = 126$ downloads. Total: $60 + 180 + 126 = 366$.

MATH500.

Llama 1B→8B — Counting & Probability, Level 5

Problem: Pat is to select six cookies from a tray containing only chocolate chip, oatmeal, and peanut butter cookies. There are at least six of each of these three kinds of cookies on the tray. How many different assortments of six cookies can be selected?

SMC-SD: We can use combinations with repetition. Distributing 6 identical items into 3 distinct groups: $C(n+k-1,k) = C(3+6-1,6) = C(8,6) = 8!/(6! \times 2!) = 28$.

Qwen 3B→14B — Algebra, Level 4

Problem: Evaluate $\lceil (3.6)^2 \rceil - (\lceil 3.6 \rceil)^2$.

SMC-SD: $(3.6)^2 = 12.96$, $\lceil 12.96 \rceil = 13$, $\lceil 3.6 \rceil = 4$, $4^2 = 16$, $13 - 16 = -3$.

AlpacaEval.

Llama 1B→8B — Geography

Instruction: Where is Indonesia?

SMC-SD: Indonesia is an archipelago of over 17,000 islands located in Southeast Asia. It is situated in the Pacific Ocean and is the world's largest archipelago, both in terms of the number of islands and land area. Indonesia is bordered by the Pacific Ocean to the east, the Indian Ocean to the south, and the South China Sea to the west. It also borders Malaysia and Brunei to the north, and the Philippines to the east. Indonesia is the world"s fourth most populous country, with over 273 million people, and its capital and largest city is Jakarta.

Llama 1B→8B — Travel math

Instruction: If it takes 8 hours to fly to Singapore, stop over for an hour and a half and then fly 14 hours to London, what is the total travel time if the plane is delayed in Singapore?

SMC-SD: To calculate the total travel time, let's break down the journey: Flight from the USA to Singapore: 8 hours. Stopover in Singapore: 1.5 hours. Flight from Singapore to London: 14 hours. Total travel time: $8 + 1.5 + 14 = 23.5$ hours. However, you mentioned the plane is delayed in Singapore. Let's assume the delay is 1 hour… Total with delay: 24.5 hours.

Llama 1B→8B — Announcement tweet

Instruction: Write an announcement tweet for medium.com readers about the new blogpost on "Open Assistant is open source ChatGPT that you don't wanna miss out".

SMC-SD: "EXCLUSIVE Launch Day Alert! Don't miss out on the surprise of the year! Our brand new blog post is LIVE on Medium: 'Open Assistant is open source ChatGPT that you don't wanna miss out'. Read now and discover the future of conversational AI! [link] #OpenAssistant #ChatGPT #AI"

Qwen 3B→14B — Argentina

Instruction: What language do Argentina people speak?

SMC-SD: The Argentinian people speak Spanish, which is the official language of Argentina. However, in some regions, there may be smaller dialects or local languages that are spoken, but they are not as widely used or respected as Spanish.

Qwen 3B→14B — Assess credibility

Instruction: How can you assess the credibility of a source of information, such as a news article or blog post, without relying solely on the reputation of the author or publisher?

SMC-SD: Assessing the credibility of a source of information, including a news article or blog post, is a matter of taking a few key steps to understand the reliability and quality of the information presented. Here"s a checklist of steps that can be followed to evaluate the credibility of the source:

- **Check the Author and Publisher's Background:** Look for the author's credentials, expertise, and the publication's reputation. A reputable author or publisher is less likely to have a history of making false claims or spreading misinformation.
- **Review the Bias and Methodology:** Assess the sources of information used in the article or blog post. Check if the data or other sources are clearly stated and verified. Look for indications of bias—bias can come from multiple sources, leading to an incomplete or potentially misleading narrative.
- **Check for Reproducibility of the Data:** Determine whether the data used in the article or blog post can be reproduced. This is crucial to ensure that the information presented is verifiable. If the data could be easily modified, then the information is less credible.
- **Evaluate the Relevance to the Topic:** The article should be based on issues of relevance and current importance. The information should address a specific problem or discuss an issue of public interest. The author must also support their claims with facts and evidence.
- **Check for Peer Reactions and Expert Consensus:** The credibility of the source also depends on the extent to which other experts in the field agree with the information. Support from respected figures can significantly bolster the credibility of a source.
- **Use External References and Other Sources:** Look for credible references that support the claims made in the article or blog post. Cross-referencing can help verify the accuracy of the information presented.
- **Look for a Balanced View:** Ensure that the article or blog post provides a fair and balanced view of the topic. Avoid articles that are overly negative or sensationalistic.

- **Check for Up to Date Information:** Keep up with the latest developments in the field or topic being discussed. This can often provide the most reliable and up-to-date information.

By following these steps, you can effectively assess the credibility of a source of information, ensuring that you obtain accurate and trustworthy information.

Qwen 3B→14B — Music theory subtopics

Instruction: Create a list of subtopics for the given topic: Music theory.

SMC-SD:

- Fundamental Concepts in Music Theory
- Rhythm and Timing in Music Theory
- Tonality and Key Systems in Music Theory
- Chord and Harmony in Music Theory
- Ear Training and Musical Sensation in Music Theory
- Music Instrumentation and Improvisation in Music Theory
- Music Analysis and Composition in Music Theory
- Genre and Style in Music Theory
- Historical Analysis of Music Theory
- Theory of Scale and Melody in Music Theory

DS1000.

Llama 1B→8B — Matplotlib, Problem 511

Prompt: Given arrays `x` and `y`, plot `x` vs. `y` and label the curve using `"x-y"` in the legend.

SMC-SD:

```
import numpy as np
import matplotlib.pyplot as plt

# Define the arrays
x = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
y = np.array([10, 9, 8, 7, 6, 5, 4, 3, 2, 1])

# Create the plot
plt.plot(x, y, label='x-y')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Plot of x vs. y')
plt.legend()

# Show the plot
plt.show()
```

Qwen 3B→14B — NumPy, Problem 292

Prompt: I want to figure out how to remove `nan` values from my array. For example, my array looks something like `x = [1400, 1500, 1600, nan, nan, nan, 1700]`. How can I remove the `nan` values from `x` to get something like `[1400, 1500, 1600, 1700]`?

SMC-SD:

```
import numpy as np

# Given array with nan values
x = np.array([1400, 1500, 1600, np.nan, np.nan, np.nan, 1700])

# Filter out nan values
x_clean = x[~np.isnan(x)]

print(x_clean)
```

K UNBIASED ESTIMATION OF TARGET-MODEL EXPECTATIONS

The weighted particle population produced by SMC-SD can also be used to estimate expectations under the target model.

Unlike previous SMC-based decoding methods Loula et al. (2025); Lew et al. (2023), weighted particle population produced by SMC-SD can be used to estimate unbiased expectations under the target model. For a fixed prompt and generation horizon, let $f(\boldsymbol{x})$ be a sequence statistic, such as target log-likelihood, sequence length, or an indicator for a histogram bin. Retaining the SMC estimate of the normalizing constant makes it possible to construct an unbiased estimator of $\mathbb{E}_p[f]$. As we show in the plot below, this estimator's variance is too high to be practically useful.

Retaining the normalizing constant. For run r, let $\overline{w}_{r,T}^{(n)}$ be the normalized terminal particle weights and $\widehat{Z}_{r,T}$ the SMC normalizing-constant estimate. For the once-per-round resampling variant in § B,

$$\widehat{Z}_{r,T} = \prod_{t=0}^{T} S_{r,t}^{N}, \qquad S_{r,t}^{N} = \frac{1}{N}\sum_{n=1}^{N} w_{r,t}^{(n)}. \tag{39}$$

With adaptive resampling, each factor instead averages the accumulated importance weights since the previous resampling boundary; the final interval contributes a factor even if it ends without resampling. The implementation accumulates these factors in log space before resetting the weights and returns $\log \widehat{Z}_{r,T}$ together with the terminal particles and their weights.

Assuming proposal support wherever the target has positive mass, correct importance weights, and conditionally unbiased resampling, the standard SMC identity gives

$$\mathbb{E}\left[\widehat{Z}_{r,T}\sum_{n=1}^{N}\overline{w}_{r,T}^{(n)} f(\boldsymbol{x}_{r,T}^{(n)})\right] = \mathbb{E}_{p}[f], \qquad \mathbb{E}[\widehat{Z}_{r,T}] = 1, \tag{40}$$

for integrable f, since the language-model target is normalized (Naesseth et al., 2024, Theorem 2.3.1). Across R independent runs, define

$$\widehat{\mu}_R(f) = \frac{1}{R}\sum_{r=1}^{R} H_r(f), \qquad H_r(f) = \widehat{Z}_{r,T}\sum_{n=1}^{N}\overline{w}_{r,T}^{(n)} f(\boldsymbol{x}_{r,T}^{(n)}). \tag{41}$$

Then $\mathbb{E}[\widehat{\mu}_R(f)] = \mathbb{E}_{p_T}[f]$. The factor $\widehat{Z}_{r,T}$ is essential: using only the normalized terminal weights generally gives a biased estimator at finite N.

Histogram construction. Let $\ell(\boldsymbol{x})$ denote the total target log-likelihood of a generated sequence. Substituting $f_B(\boldsymbol{x}) = \mathbb{1}\{\ell(\boldsymbol{x}) \in B\}$ in (41) yields an unbiased estimate of the target probability mass in bin B. Figure 9 shows the unbiased distribution under the target distribution using the unbiased estimator given by SMC-SD. The bars are divided by the number of runs, without renormalizing their total weight. For bins covering the full support, their total mass is $R^{-1}\sum_r \widehat{Z}_{r,T}$ and need not equal one in a finite experiment. For several prompts, the same construction applies separately to each prompt and can be averaged with fixed prompt weights. Uncertainty is assessed over independent runs, since particles within a run share resampled ancestors. Unbiasedness does not guarantee low variance, however. Fluctuations in the normalizing-constant estimate $\widehat{Z}_{r,T}$ can make $H_r(f)$ highly variable across runs, requiring many independent runs for reliable estimates. The saved histogram reports an estimated standard error of 8.05 for SMC-SD, compared with 0.49 for direct target sampling, indicating substantially greater reported Monte Carlo uncertainty for the SMC-SD mean. These standard errors quantify uncertainty in the estimated means; comparing the variances of individual runs would also require the run counts.

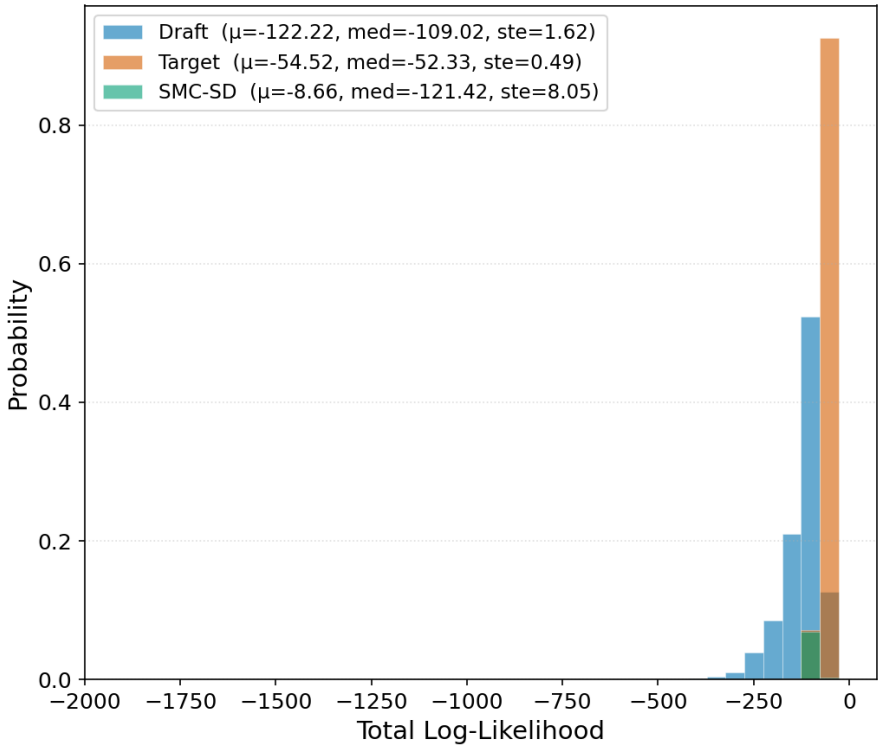


Figure 9: Saved exploratory log-likelihood histogram comparing draft, target, and SMC-SD estimates. The vertical axis reports probability mass per bin. SMC-SD retains the normalizing-constant factor in (41); its bars therefore need not sum to one. The original legend reports the mean (μ), weighted median (med), and estimated standard error (ste). The display is restricted to log-likelihoods above -2000.