PRESTO: PREDICTIVE SUBTREE PREFETCHING FOR FAST LLM POWER SAMPLING

Anonymous authors
Paper under double-blind review

ABSTRACT

Power sampling improves large language model (LLM) reasoning at test time by sampling from a sharpened distribution over the base model. Its sampler, PowerMH, is slow because its Metropolis–Hastings (MH) updates are sequential: each accept/reject decision sets the state for the next proposal. Prior work cuts this cost by changing the sampler or the model (e.g., sequential Monte Carlo, approximate lookahead, entropy-guided proposals, distillation). We instead keep the sampler: Predictive SubTree Prefetching (PreSTO) completes multiple MH transitions per model call without changing the MH transition kernel. Future MH outcomes form a tree of accept/reject decisions; PreSTO builds a *prefetching subtree* of it, generates every proposal whose prefix is already available in one batched call, and follows the realized path with the original MH acceptance rule. Because MH reaches only a sparse, nearly unary part of each prefetched tree, we choose which decisions to prefetch under a budget by dynamic programming. On PowerMH and its follow-ups EntropyCut and MultiTryMH across 7 datasets and 5 base LLMs, PreSTO achieves median speedups of $1.71\times$, $1.88\times$, and $2.45\times$, respectively; on the first two, it makes a median of $3.6\times$ fewer model calls, each only $2.1\times$ longer. Our predictive traversal rule improves the mean speedup over the default breadth-first traversal.

1 INTRODUCTION

Large language models (LLMs) perform strongly on reasoning tasks such as competition mathematics, code generation, and graduate-level science questions (Hendrycks et al., 2021b; Li et al., 2022; Rein et al., 2024). These capabilities are commonly improved by reinforcement learning (RL) post-training with task-specific rewards (Guo et al., 2025; Hu et al., 2025). Recent work suggests that RL post-training largely sharpens the base model's distribution (He et al., 2025; Shao et al., 2026; Chen et al., 2025), and that sampling directly from a sharpened distribution, known as *power sampling*, can perform comparably to RL post-training without additional training (Karan & Du, 2026).

However, PowerMH, the Metropolis–Hastings (MH) sampler for power sampling, can cost up to $8.8\times$ as much as standard sampling (Ji et al., 2026; Karan & Du, 2026). Follow-up work improves it in four directions: (1) *Alternative samplers* replace serial MH with sequential Monte Carlo (SMC) or replica exchange (Azizi et al., 2026; He et al., 2026). (2) *Efficient approximations* approximate the power target to reduce inference cost (Ji et al., 2026; Nguyen et al., 2026). (3) *Proposal refinements* choose cut indices by token-level entropy to improve mixing (Zhou et al., 2026; Guo et al., 2026). (4) *Objective or training changes* target the answer marginal (Arzhantsev et al., 2026) or fine-tune the model on power-sampled responses (Tomihari & Sato, 2026).

The main bottleneck of PowerMH is the sequential dependence between MH transitions: in a standard implementation, each proposal must be accepted or rejected before the next is generated from the resulting state. We draw on prefetching in classical MCMC (Brockwell, 2006; Angelino et al., 2014): future MH transitions form a tree of accept/reject outcomes, and each sequential run traces one root-to-leaf path. If the proposals on this path are available in advance, several transitions complete without another proposal-model call.

We introduce Predictive SubTree Prefetching (PreSTO), a framework for MH-based sharpened sampling, and its PowerMH instantiation, PreSTO-PowerMH. In each round, PreSTO builds a *prefetching subtree*, generates its feasible proposals in one batched proposal-LLM call, and follows the

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.

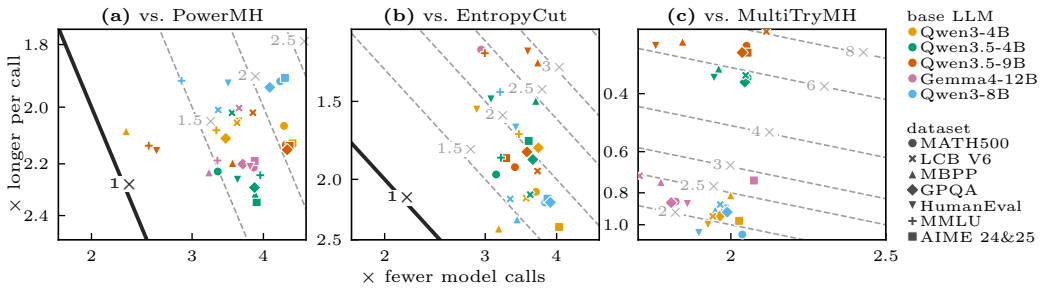


Figure 1: Across 92 sampler–LLM–dataset settings, PreSTO (ours) speeds up PowerMH, EntropyCut, and MultiTryMH by a median of $1.71\times$, $1.88\times$, and $2.45\times$, respectively.

realized path with the original MH acceptance rule. We prove that PreSTO-PowerMH leaves the MH transition kernel of PowerMH unchanged (Proposition 1). Because MH reaches only a sparse, nearly unary part of each prefetched tree, we propose a *predictive traversal rule* that allocates the prefetch budget by dynamic programming (Section 3.3). With prefix-cache reuse across requests, these mechanisms reduce total latency whenever the savings from fewer serial calls outweigh the extra cost of each batch.

We apply PreSTO to PowerMH, EntropyCut, and MultiTryMH on seven datasets and five base LLMs (Figure 1). PreSTO achieves median speedups of $1.71\times$ over PowerMH and $1.88\times$ over EntropyCut (up to $2.29\times$ and $2.85\times$), and $2.45\times$ over MultiTryMH, whose timings differ in scope (Table 1). Case studies show that **(1)** the log-likelihood and confidence of PreSTO outputs pass paired equivalence tests against the baselines in 29 of 34 comparisons (Figure 6); **(2)** the extra memory stays under 7% of the preallocated KV-cache pool (Table 2); and **(3)** the predictive traversal rule achieves a higher mean speedup than the default breadth-first (BFS) traversal at all six prefetch budgets tested on LCB V6 (Figure 7).

2 Preliminaries

Metropolis–Hastings (MH) is an MCMC algorithm for sampling from a distribution known only up to a normalizing constant (Metropolis et al., 1953; Hastings, 1970). MH constructs a Markov chain $\mathbf{x}^{(0)} \ldots \to \mathbf{x}^{(k)} \cdots \to \mathbf{x}^{(K)}$ to sample from a target distribution $\pi(\mathbf{x})$. Given current state $\mathbf{x}^{(k)}$, a candidate $\mathbf{y}$ is drawn from a proposal distribution $q(\cdot \mid \mathbf{x}^{(k)})$ and is accepted with probability

$$A(\mathbf{y}, \mathbf{x}^{(k)}) = \min\left\{1, \frac{\pi(\mathbf{y})\, q(\mathbf{x}^{(k)} \mid \mathbf{y})}{\pi(\mathbf{x}^{(k)})\, q(\mathbf{y} \mid \mathbf{x}^{(k)})}\right\}. \tag{1}$$

If the proposal is accepted, the chain moves to $\mathbf{x}^{(k+1)} \leftarrow \mathbf{y}$; otherwise, $\mathbf{x}^{(k+1)} \leftarrow \mathbf{x}^{(k)}$.

Power Sharpening for LLMs. Let $\mathbf{x}_0$ denote a given prompt and $\mathbf{x}_{0:n} = (\mathbf{x}_0, x_1, \ldots, x_n)$ be a sequence of generated tokens appended to that prompt. Let p_0 denote the distribution of the base LLM. Recently, Karan & Du (2026) propose to sample from a *power-sharpened* distribution:

$$\pi_\alpha(\mathbf{x}_{1:n} \mid \mathbf{x}_0) \;\propto\; p_0(\mathbf{x}_{1:n} \mid \mathbf{x}_0)^\alpha. \tag{2}$$

When $\alpha > 1$, the exponent sharpens the base distribution to amplify high-likelihood continuations and suppress low-likelihood ones (Li et al., 2025). Empirical results (Karan & Du, 2026; Ji et al., 2026) show that power sharpening can achieve performance competitive with RL-based post-training methods, such as Group Relative Policy Optimization (GRPO) (Shao et al., 2024).

Exact sampling from π_α in equation 2 is intractable, as it requires summing over future continuations. Prior work (Karan & Du, 2026) proposes PowerMH, an MH-based sampler that proceeds through the sequence of intermediate targets block by block:

$$\pi_\alpha(\mathbf{x}_{1:B} \mid \mathbf{x}_0) \to \cdots \to \pi_\alpha(\mathbf{x}_{1:\ell B} \mid \mathbf{x}_0) \to \cdots \to \pi_\alpha(\mathbf{x}_{1:n} \mid \mathbf{x}_0), \tag{3}$$

where B is the block size. At the ℓ-th stage, PowerMH runs MH steps targeting the distribution $\pi_\alpha(\mathbf{x}_{1:\ell B} \mid \mathbf{x}_0)$ by first extending the sample $\mathbf{x}_{0:(\ell-1)B}$ from the previous stage by one block to produce $\mathbf{x}_{1:\ell B}$, and then refining it with several MH steps.

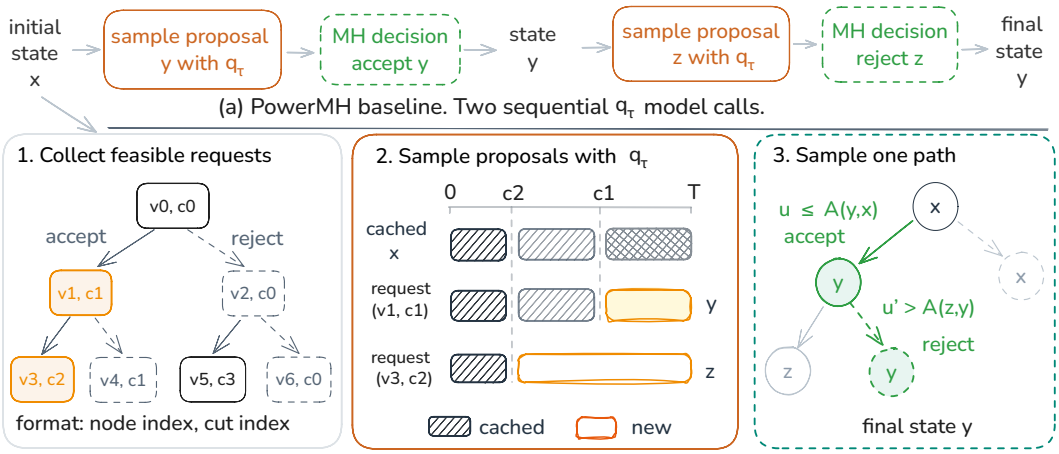


Figure 2: Our subtree prefetching accelerates PowerMH for trajectory $\mathbf{x} \to \mathbf{y} \to \mathbf{y}$. **(a)** PowerMH performs two sequential rounds of proposal generation with q_τ followed by an MH accept/reject decision. **(b)** Our method collects feasible requests over the prefetching tree, generates proposals $\mathbf{y}, \mathbf{z}$ in one batched call, and then samples a path over the tree.

3 Methodology

3.1 Motivation

Prior studies (Ji et al., 2026; Azizi et al., 2026) show that PowerMH incurs substantial inference-time latency. On the MATH500 dataset, Karan & Du (2026) report that PowerMH is up to $8.8\times$ slower than standard sampling. Follow-up variants (Zhou et al., 2026; Guo et al., 2026) incur even greater latency. This latency arises from the *inherently sequential nature of MH*: the proposal at step $k{+}1$ can be generated only after the accept/reject outcome at step k determines the next state.

We first describe the suffix-resampling procedure in the PowerMH sampler. For notation clarity, let $L = (\ell - 1)B$ and $T = \ell B$ be the sequence length before and after the ℓ-th stage. Given an initial sequence $\mathbf{x}^{(0)} = \mathbf{x}_{0:L}$, the sampler first extends $\mathbf{x}_{0:L}$ to $\mathbf{x}_{0:T}$ by sampling a continuation from the proposal LLM. At step k with MH state $\mathbf{x}^{(k)}$, the sampler draws a cut index $c \sim \mathrm{Unif}\{0, \ldots, T-1\}$, and resamples the suffix from the proposal LLM to construct candidate $\mathbf{y} = \left(\mathbf{x}^{(k)}_{0:c}, \mathbf{y}_{c+1:T}\right)$:

$$y_t \sim q_\tau\left(\cdot \mid \mathbf{x}^{(k)}_{0:c}, \mathbf{y}_{c+1:t-1}\right), \qquad \text{for } t = c+1, \ldots, T, \tag{4}$$

where the proposal distribution q_τ is the base LLM with temperature τ. The next state $\mathbf{x}^{(k+1)}$ is then determined by applying the acceptance rule equation 1 to the candidate $\mathbf{y}$.

MCMC Prefetching is a form of *speculative computation* in MCMC that accelerates an inherently sequential process by precomputing multiple possible future MH transitions before the corresponding accept/reject decisions are realized (Brockwell, 2006; Angelino et al., 2014; Ye & Lu, 2024). In our setting, prefetching is best understood through a depth-K tree, which enumerates all possible accept/reject outcomes over the next K MH steps. Each tree node v_i is associated with a cut index c_i and a proposed candidate. Its left branch corresponds to accepting a newly proposed candidate, whereas its right branch corresponds to a rejection decision. Every realization of K MH transitions corresponds to a root-to-leaf path in the tree. Example 1 explains this construction in detail.

Example 1. Figure 2(a) shows the two-step PowerMH trajectory $\mathbf{x} \to \mathbf{y} \to \mathbf{y}$. In round 1, starting at $\mathbf{x}$, the sampler draws cut index c_1, resamples the suffix according to equation 4 to generate $\mathbf{y}$, and accepts the proposal, yielding $\mathbf{x}^{(1)} = \mathbf{y}$. In round 2, it draws a new cut index c_2 and resamples the suffix of $\mathbf{y}$ to generate $\mathbf{z}$, but rejects this proposal, leaving $\mathbf{x}^{(2)} = \mathbf{y}$. These accept/reject decisions select the path $\mathbf{x} \to \mathbf{y} \to \mathbf{y}$ in the prefetched tree in step 3 of Figure 2(b).

To accelerate PowerMH and its follow-up variants, we propose to construct a *prefetching subtree* that represents multiple possible future MH states and then selects one root-to-leaf path using equation 1.

3.2 SUBTREE PREFETCHING FOR MH-BASED POWER SHARPENING

Our method is composed of (1) collecting feasible suffix-resampling requests from the subtree, (2) sampling proposal sequences and computing acceptance probabilities, and (3) performing MH transitions over the prefetched subtree by moving to the left or right child recursively.

(1) Collecting feasible requests. We use breadth-first search (BFS) to traverse the prefetching tree, to collect at most N_{pf} feasible suffix-resampling requests. In the BFS construction, we initialize a queue at the subtree root and traverse nodes in breadth-first order. Each request is a tuple (v_i, c_i), where v_i identifies the node and c_i denotes its cut index. Expanding a node creates two children: an acceptance (left) child and a rejection (right) child. For each acceptance child v_i, we draw a new cut index $c_i \sim \text{Unif}\{0, \ldots, T-1\}$ and form the request (v_i, c_i). If this request satisfies Condition 1, we add it to the batch and enqueue v_i for further expansion. Otherwise, we defer the request while retaining its original cut index; if its branch is reached, we evaluate it in a later batch. We enqueue the rejection child but do not add a corresponding request to the batch, because the child retains its parent's state and therefore requires no new proposal. The traversal stops when the batch contains N_{pf} requests or the queue is empty. Section 3.3 introduces our predictive traversal rule, which improves on the default BFS traversal by maximizing the expected-transition objective, informed by an empirical property of prefetching trees observed across all datasets in Figure 3.

Condition 1. *Let v be a node in the prefetching tree with cut index c and v' be its acceptance child with cut index c'. The request associated with v' is feasible for the current prefetch batch if $c' \leq c$.*

Condition 1 follows from the suffix-resampling step in equation 4. Fix a state $\mathbf{x} = \mathbf{x}_{0:T}$ and a cut $0 \leq c \leq T-1$, and let $\mathbf{y} = (\mathbf{x}_{0:c}, \mathbf{y}_{c+1:T})$ be a candidate obtained by suffix resampling at c. For any $0 \leq c' \leq c$, the preserved prefix satisfies $\mathbf{y}_{0:c'} = \mathbf{x}_{0:c'}$. Hence, for every suffix $\mathbf{z}_{c'+1:T}$, autoregressive factorization gives

$$q_\tau(\mathbf{z}_{c'+1:T}|\mathbf{y}_{0:c'}) = \prod_{t=c'+1}^{T} q_\tau(z_t|\mathbf{y}_{0:c'}, \mathbf{z}_{c'+1:t-1}) = \prod_{t=c'+1}^{T} q_\tau(z_t|\mathbf{x}_{0:c'}, \mathbf{z}_{c'+1:t-1}) = q_\tau(\mathbf{z}_{c'+1:T}|\mathbf{x}_{0:c'}).$$

Consequently, the proposal at cut c' can be generated in the same batch from the already available prefix $\mathbf{x}_{0:c'}$, before $\mathbf{y}$ is available, without changing its conditional distribution. Example 2 visualizes how BFS collects suffix-resampling requests and checks Condition 1.

Example 2. To collect a batch of $N_{\text{pf}} = 2$ requests from the tree in step 1 of Figure 2(b), we start BFS at the root node v_0, which represents the current sequence $\mathbf{x}_{0:T}$. We assign it the cut index $c_0 = T$ for consistency with Condition 1. We expand the tree in the order $v_0, v_1, v_2, \ldots$. Whenever we visit an acceptance (left) child, we independently sample a new cut index for it. If each sampled cut index satisfies Condition 1, we add the request (v_1, c_1), followed by the request (v_3, c_2).

As shown in step 2 of Figure 2(b), resampling the suffix at v_1 after cut index c_1 leaves the prefix $\mathbf{x}_{0:c_2}$ unchanged because $c_2 \leq c_1$. Then, the proposal at v_3 is conditioned only on the shared prefix $\mathbf{x}_{0:c_2}$, which is already available at the root node v_0. The two suffix-resampling requests (v_1, c_1) and (v_3, c_2) can therefore be issued in the same proposal model call.

(2) Batched proposal sampling. The collected suffix-resampling requests are submitted in a single batched call to the serving engine of the proposal LLM (e.g., vLLM or SGLang), which returns the resampled suffixes together with their log-probabilities. These log-probabilities are used to compute the MH log-acceptance probability. Because many requests share long prefixes, the KV-cache for each shared prefix can be computed once and reused by all requests that extend it. Modern LLM serving engines support this reuse through *prefix caching*. The following Example 3 explains the detailed calculation procedure. Appendix Example 4 visualizes how KV-cache reuse reduces repeated prefix computation across batched requests.

Example 3. In steps 2 and 3 of Figure 2(b), sequence $\mathbf{x}$ is associated with the root v_0, while $\mathbf{y}$ and $\mathbf{z}$ are the newly resampled sequences associated with nodes v_1 and v_3, respectively, in the tree shown in step 3. For the request (v_1, c_1), $\mathbf{y}$ is obtained by resampling the suffix from the proposal LLM q_τ conditioned on the prefix $\mathbf{x}_{0:c_1}$. Thus, $\mathbf{y} = \big(\mathbf{x}_{0:c_1}, \mathbf{y}_{c_1+1:T}\big)$. By equation 1, the log-acceptance probability for the MH transition from $\mathbf{x}$ to $\mathbf{y}$ is

$$\begin{aligned}\log A(\mathbf{y}, \mathbf{x}) = \min\{0, &\log q_\tau(\mathbf{x}_{c_1+1:T} \mid \mathbf{x}_{0:c_1}) - \log q_\tau(\mathbf{y}_{c_1+1:T} \mid \mathbf{x}_{0:c_1}) \\ &+ \alpha\left(\log p_0(\mathbf{y}_{c_1+1:T} \mid \mathbf{x}_{0:c_1}) - \log p_0(\mathbf{x}_{c_1+1:T} \mid \mathbf{x}_{0:c_1})\right)\}.\end{aligned} \tag{5}$$

The calculations are performed in the log domain for numerical stability. Each sequence-level term is computed by summing token-level log-probabilities, such as $\log q_\tau(\mathbf{y}_{c_1+1:T} \mid \mathbf{x}_{0:c_1}) = \sum_{t=c_1+1}^{T} \log q_\tau(y_t \mid \mathbf{x}_{0:c_1}, \mathbf{y}_{c_1+1:t-1})$. Appendix B.1 provides the derivation of equation 5.

(3) MH transitions over the prefetched subtree. Starting from the subtree root, representing the current MH state, we simulate the MH accept/reject decisions sequentially along a single root-to-leaf path. At each node, we follow the left edge to v_i with the acceptance probability; otherwise, we follow the corresponding right edge. The resulting root-to-leaf path determines a sequence of accept/reject decisions and, consequently, the final MH state. We update the current state to the sequence associated with the realized leaf and retain its token-level log-probabilities under the proposal model q_τ and the base model p_0 for computing MH acceptance probabilities. Appendix Example 5 visualizes the above process with the calculated acceptance probabilities from equation 5.

Finally, Algorithms 1 and 2 compare PowerMH with PreSTO-PowerMH, illustrating how PreSTO executes multiple MH transitions. Proposition 1 further establishes that PreSTO-PowerMH preserves the original MH transition kernel.

3.3 Predictive Subtree Allocation for Efficient Prefetching

Prefetching efficiency depends on total workload to generate a batch and on the number of MH transitions that the batch completes. We show that MH reaches only a sparse, nearly unary part of the prefetched tree, and use this observation to decide which decisions to prefetch under a budget.

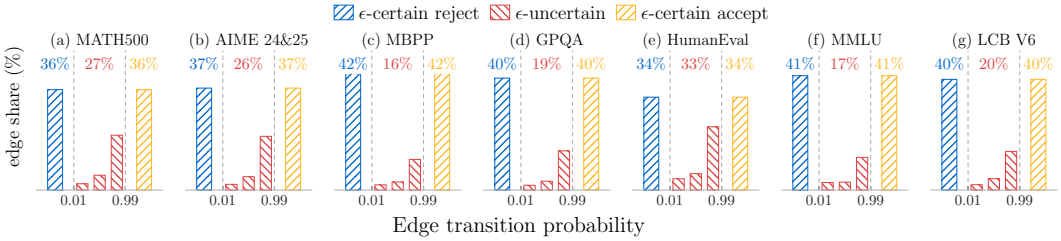


Figure 3: Across all datasets, approximately 67%–84% of edges are ϵ-certain in the prefetched trees. Appendix E.5 reports additional acceptance distributions.

Our Observation. Fix a small $\epsilon > 0$ (we use $\epsilon = 0.01$) and let u be an expanded node with acceptance child u^+ and rejection child u^-, reached with probabilities $A(u^+, u)$ and $1 - A(u^+, u)$. The decision at u is an ϵ-certain accept if $A(u^+, u) \geq 1 - \epsilon$, an ϵ-certain reject if $A(u^+, u) \leq \epsilon$, and ϵ-uncertain otherwise. Rounding the acceptance probability of an ϵ-certain decision to 1 or 0 leaves a single reachable child, whereas an ϵ-uncertain decision keeps both. Figure 3 shows that most decisions are ϵ-certain at $\epsilon = 0.01$: pooling the acceptance and rejection edges of each decision, about 80% of the edges are ϵ-certain on LCB V6 (panel (g)).

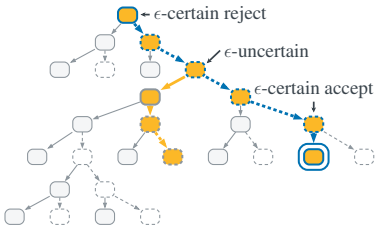


Figure 4: Orange nodes mark the ϵ-rounded subtree at $\epsilon = 0.01$; blue edges trace a sampled MH path.

Figure 4 shows the resulting reachable subtree: orange nodes remain reachable after rounding, and blue edges trace a sampled path of five MH transitions. Most of the tree lies outside this subtree, and the subtree itself is predominantly unary. Yet every expanded decision costs one proposal request, even a near-certain rejection. Because BFS expands decisions regardless of how likely MH is to reach them, it spends much of the budget on nodes that MH rarely reaches, which motivates allocating requests selectively. Appendix E.6 shows additional tree visualizations.

Predictive traversal rule. Before generation, we predict the ϵ-certainty class probabilities of each candidate decision and convert them into a branch probability; the cut position alone suffices, with held-out AUCs of 0.68–0.77 under PreSTO-PowerMH (Appendix E.4). We then select an ancestor-closed set of at most N_{pf} decisions that maximizes the predicted number of completed transitions (Lemma 1), solved exactly by a dynamic program over the acceptance and rejection subtrees (Appendix C.3). We call this the *predictive traversal rule*, in contrast to the default BFS traversal of Section 3.2.

4 Related Work

We review related work most relevant to PreSTO here; see Appendix A for an extended discussion.

MCMC prefetching. Like prior MCMC-prefetching methods (Brockwell, 2006; Angelino et al., 2014; Ye & Lu, 2024), PreSTO precomputes candidates for future accept/reject outcomes before the realized path is known. The methods differ in how they allocate this speculative work. Brockwell (2006) and Angelino et al. (2014) treat proposal generation as cheap and therefore prefetch many more candidates than the chain eventually uses. LLM proposals are expensive, so PreSTO confines prefetching to a budgeted subtree. Ye & Lu (2024) expand a regular multiproposal tree with a fixed number of proposals and a fixed prefetch depth, whereas PreSTO uses Condition 1 to identify feasible suffix requests and prioritizes them by estimated benefit and cost. Finally, multiproposal MH methods (Ye & Lu, 2024) combine several candidates within a single transition kernel. PreSTO schedules several consecutive PowerMH transitions while leaving each transition kernel unchanged.

Tree-based speculative decoding. Tree-based speculative decoding (TreeSD) (Stern et al., 2018; Leviathan et al., 2023; Chen et al., 2023; Miao et al., 2024) also batches evaluations over a tree to reduce serial model calls, but its tree encodes different objects. In TreeSD, branches are alternative *draft-token continuations*, and target-model verification commits an accepted token prefix (Cai et al., 2024; Li et al., 2024b;a). In PreSTO, branches are the accept/reject outcomes of *complete-sequence MH proposals*, and following the realized MH decisions reproduces the sequential PowerMH trajectory. The shared tree structure admits similar budget-allocation arguments, but the unit of progress differs: each visited expanded node in PreSTO advances the chain by one MH transition, whether the proposal is accepted or rejected.

Power-sharpening methods. Other work targets the power distribution through SMC-based samplers (Azizi et al., 2026; He et al., 2026), efficient approximations (Ji et al., 2026; Nguyen et al., 2026), refined proposals (Zhou et al., 2026; Guo et al., 2026), or modified objectives and training procedures (Arzhantsev et al., 2026; Tomihari & Sato, 2026). PreSTO takes a different route: it accelerates PowerMH itself, keeping its target, suffix proposal, and MH acceptance rule intact, and changes only the schedule by which proposals are generated and scored.

5 Experiments

In this section, we show that PreSTO speeds up both samplers across base LLMs and datasets, and use case studies to examine where the speedup comes from, whether the outputs are preserved, and how the prefetch budget and traversal rule affect the speedup. Appendix D describes the baselines, base LLMs, datasets, and metrics, and Appendix E reports the extended results.

5.1 Acceleration across base LLMs and datasets

Speedup over sequential samplers. We evaluate whether PreSTO lowers the execution cost of MH-based power sampling without changing its output. We pair each sequential sampler, PowerMH (Karan & Du, 2026), EntropyCut (Zhou et al., 2026), and MultiTryMH (Guo et al., 2026), with its own PreSTO implementation. Both samplers in a pair run for $K = 100$ realized MH transitions on the first 20 prompts of each dataset, generating the full $1{,}024$-token response as a single block on the same vLLM backend, with the base LLM, target exponent, proposal rule, initialization, and prompts held fixed (Appendix D.6).

Table 1 summarizes relative speedups within each available model–dataset pair. For PowerMH (Table 1a), the four Qwen models reach $1.00\times$–$2.29\times$, with at least $1.5\times$ in 23 of their 28 available settings, and Gemma4-12B-it reaches $1.44\times$–$1.81\times$, with at least $1.5\times$ in 6 of its 7 settings. For EntropyCut (Table 1b), PreSTO-EntropyCut is faster in all 29 available settings, reaching $1.32\times$–$2.85\times$ and at least $1.5\times$ in 27. The speedup equals the reduction in the number of model calls divided by the increase in time per call equation 23. Under PowerMH at $N_{\mathrm{pf}} = 20$ (all datasets except LCB V6), Gemma4-12B-it realizes 3.40–3.96 transitions per call, but each call takes about $2.2\times$ as long as a PowerMH step, which limits its speedup to $1.44\times$–$1.76\times$. A slower sequential step leaves more room: on MBPP with Qwen3.5-9B, an EntropyCut step takes 7.62 seconds versus 4.97 for PowerMH, so PreSTO-EntropyCut makes $3.71\times$ fewer calls at only $1.30\times$ the per-call latency and reaches $2.85\times$, against $1.61\times$ for PreSTO-PowerMH. Neither sampler benefits uniformly more:

Table 1: Speedup of PreSTO over sequential samplers across base LLMs and datasets.

(a) PowerMH vs. PreSTO-PowerMH

Base LLM	MATH500	AIME 24&25	MBPP	GPQA	HumanEval	MMLU	LCB V6
Qwen3-4B	2.11×	2.12×	1.10×	1.63×	1.77×	1.59×	1.75×
Qwen3-8B	2.23×	**2.29**×	1.00×	2.12×	1.81×	1.50×	1.66×
Qwen3.5-4B	1.49×	1.66×	1.67×	1.69×	1.59×	1.76×	1.75×
Qwen3.5-9B	2.06×	2.08×	1.61×	2.05×	1.21×	1.18×	1.90×
Gemma4-12B-it	1.74×	1.76×	1.44×	1.67×	1.71×	1.52×	1.81×

(b) EntropyCut vs. PreSTO-EntropyCut

Base LLM	MATH500	AIME 24&25	MBPP	GPQA	HumanEval	MMLU	LCB V6
Qwen3-4B	1.76×	1.69×	1.32×	2.09×	1.88×	2.03×	1.65×
Qwen3-8B	1.75×	1.80×	1.47×	1.79×	2.05×	2.20×	1.54×
Qwen3.5-4B	1.60×	2.06×	2.46×	1.96×	2.06×	1.73×	1.70×
Qwen3.5-9B	1.77×	1.76×	**2.85**×	1.96×	**2.85**×	2.38×	1.91×

(c) MultiTryMH vs. PreSTO-MultiTryMH

Base LLM	MATH500	AIME 24&25	MBPP	GPQA	HumanEval	MMLU	LCB V6
Qwen3-4B	–	2.08×	2.46×	2.09×	1.94×	–	2.07×
Qwen3-8B	1.90×	2.22×	2.17×	2.17×	1.80×	–	2.27×
Qwen3.5-4B	5.72×	–	5.85×	5.53×	5.46×	–	5.81×
Qwen3.5-9B	7.15×	6.82×	6.70×	6.77×	6.27×	–	**8.13**×
Gemma4-12B-it	2.17×	2.82×	2.44×	2.14×	2.17×	–	2.47×

in the 27 Qwen settings in which both samplers use the same problem set, the EntropyCut speedup is larger in 15, including all four Qwen models on MBPP and HumanEval. The EntropyCut comparison covers fewer settings: it uses the base Gemma4-12B checkpoint, for which only MATH500 has complete runs of both samplers. For MultiTryMH, PreSTO-MultiTryMH reduces the mean total time of the sequential sampler by $1.8\times$–$8.1\times$ at $K = 100$, partly because its implementation skips candidate rescoring (Table 8). Appendix E.1 reports further execution results.

5.2 Case Studies

Case Study 1: Model-call efficiency. We break the speedup into cumulative model-call time (sec), realized MH transitions per model call, and time per model call (sec/call); the latter two characterize chain progress and cost per call. Figure 5 reports these metrics on LCB V6 with Qwen3.5-9B, and Figure 1 places each of the 92 available sampler–LLM–dataset settings by the two factors of the speedup in equation 23: the reduction in the number of model calls and the increase in time per call.

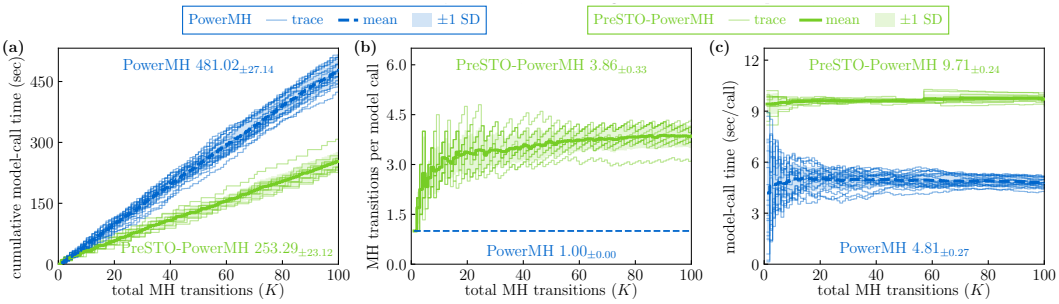


Figure 5: Under LCB V6 and Qwen3.5-9B ($N_{\text{pf}} = 10$), PreSTO speeds up PowerMH by $1.90\times$ in cumulative model-call time: it makes $3.83\times$ fewer model calls, each taking $2.02\times$ as long.

From Figure 5, we find that **(1)** PreSTO-PowerMH completes $K = 100$ transitions in $253.29_{\pm 23.12}$ seconds versus $481.02_{\pm 27.14}$ for PowerMH, a $1.90\times$ speedup. **(2)** Each PreSTO-PowerMH call realizes $3.86_{\pm 0.33}$ transitions instead of one, but takes $9.71_{\pm 0.24}$ rather than $4.81_{\pm 0.27}$ seconds. **(3)** EntropyCut shows the same pattern (Table 1(b)): PreSTO-EntropyCut reduces cumulative time from $463.80_{\pm 52.32}$ to $242.89_{\pm 35.93}$ seconds ($1.91\times$), with $4.22_{\pm 0.58}$ transitions per call at $9.00_{\pm 0.49}$

rather than $4.64_{\pm 0.52}$ seconds per call. **(4)** Across the 92 settings (Figure 1), the speedup over PowerMH follows the reduction in calls (1.94–4.50×; 95% of the variance in log speedup) because the increase in time per call varies little (1.90–2.35×); the speedup over EntropyCut follows the increase in time per call (1.24–2.40×; nearly all of the variance) because the reduction in calls is nearly constant (2.90–4.04×). Over MultiTryMH, the reduction in calls is again nearly constant (1.75–2.10×), and the speedup follows the time per call (0.26–1.07× that of a MultiTryMH step; 97% of the variance), consistent with MultiTryMH rescoring its candidates while PreSTO-MultiTryMH does not (Table 8). Overall, 84 of the 92 settings reach at least 1.5×; Appendix E.1 reports the remaining datasets and base LLMs.

Case Study 2: Effect of prefetch budget. We sweep $N_{\text{pf}} \in \{4, 8, 12, 16, 20\}$ for PreSTO-PowerMH with the default BFS traversal on LCB V6 with Qwen3.5-9B, running PowerMH and PreSTO-PowerMH with the same preallocated KV-block pool (1.62M tokens). Table 2 reports chain progress (transitions per call, speedup over PowerMH), uncached workload (prefix-cache hit rate, miss tokens per transition), and memory (peak KV-pool use); each budget uses a single seed.

Table 2: Under dataset LCB V6 and base LLM Qwen3.5-9B, raising the prefetch budget N_{pf} from 4 to 16 increases the speedup of PreSTO-PowerMH over PowerMH from 1.42× to 1.94× while occupying at most 5.01% of the preallocated KV pool.

	PowerMH	PreSTO-PowerMH, prefetch budget N_{pf}				
		4	8	12	16	20
MH transitions per model call	1.00	2.45	3.33	3.74	4.25	4.12
Speedup over PowerMH	1.00×	1.42×	1.72×	1.83×	1.94×	1.83×
Prefix-cache hit rate (%)	73.1	72.9	71.6	71.4	71.2	70.7
Prefix-cache miss tokens per transition (10^3)	0.28	0.43	0.64	0.84	0.97	1.24
Peak KV-pool use (%)	0.51	1.67	2.57	3.98	5.01	6.68

From Table 2, we find that **(1)** The speedup grows with the budget up to $N_{\text{pf}} = 16$ and then saturates: from $N_{\text{pf}} = 4$ to 16, transitions per call rise from 2.45 to 4.25 and the speedup from 1.42× to 1.94×, while at $N_{\text{pf}} = 20$ both fall slightly (4.12, 1.83×). **(2)** The speedup comes from more transitions per call, not from cache reuse: the prefix-cache hit rate stays at 70.7–72.9% (PowerMH: 73.1%), while each call realizes 2.45–4.25 transitions in 8.3–10.8 seconds, versus one transition in 4.8 seconds for PowerMH. **(3)** The extra memory is negligible, but the uncached work is not. Peak pool use rises from 0.51% to at most 6.68%, and since vLLM reserves the pool at startup, peak GPU memory is 75.26 GiB for both methods (within 4 MiB); miss tokens per transition, however, grow 1.5–4.4× over PowerMH because every prefetched proposal is generated whether or not it is realized. Thus $N_{\text{pf}} \in [12, 16]$ gives most of the speedup at 4–5% pool use.

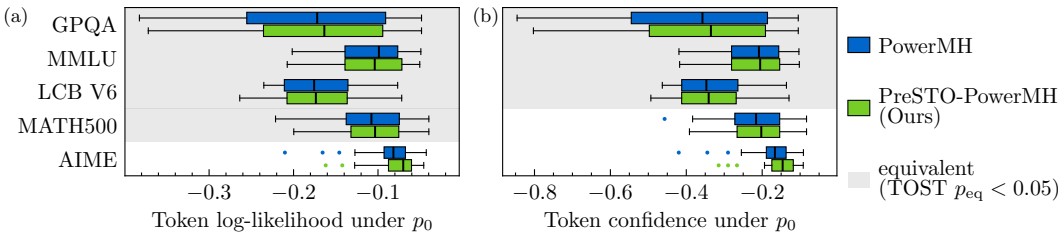


Figure 6: Under base LLM Qwen3.5-4B across five datasets, PreSTO-PowerMH and PowerMH produce closely overlapping distributions of base-model log-likelihood and confidence, and the paired TOST finds them equivalent in 7 of the 10 comparisons (shaded).

Case Study 3: Likelihood and confidence. PreSTO changes how proposals are scheduled and evaluated, not the MH target, so a sampler and its PreSTO variant should produce terminal traces with the same distribution of token-normalized log-likelihood and confidence under p_0 (Karan & Du, 2026). We rescore the terminal traces of each pair under p_0 after $K = 100$ transitions (Appendix E.3). We test each pair with a paired TOST equivalence test at a margin of 0.2 baseline standard deviations.

Figure 6 shows the per-prompt distributions for PowerMH and PreSTO-PowerMH on five datasets with Qwen3.5-4B. The two samplers' distributions overlap closely on every dataset, and the test establishes equivalence in 7 of the 10 comparisons: log-likelihood on MATH500, GPQA, MMLU, and LCB V6, and confidence on GPQA, MMLU, and LCB V6. The remaining three, both statistics on AIME and confidence on MATH500, have 90% intervals that extend past the margin. Across all base LLM–dataset pairs that pass the rescoring check of Appendix E.3 (8 for PowerMH and 9 for EntropyCut, two statistics each), 29 of the 34 tests establish equivalence, and the pooled differences on both statistics are equivalent to zero ($p_{\text{eq}} < 0.001$; Appendix E.3).

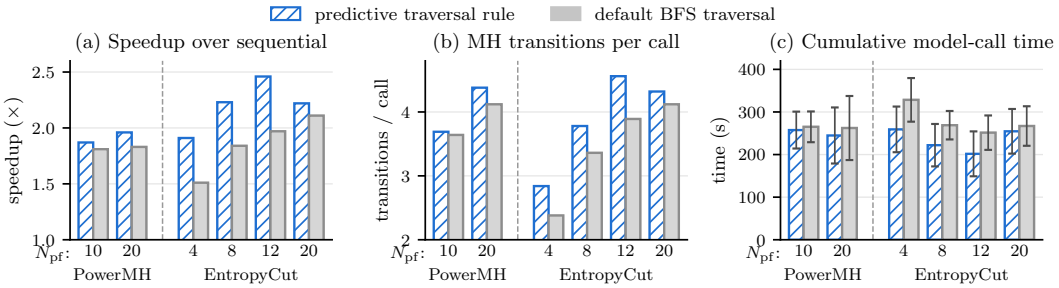


Figure 7: Under LCB V6 and Qwen3.5-9B, replacing the default BFS traversal with the predictive traversal rule raises the mean speedup of PreSTO over the sequential sampler at every prefetch budget, from 1.51–2.11× to 1.87–2.46×.

Case Study 4: Effect of traversal rules. We compare the default BFS traversal with the predictive traversal rule, which solves equation 20 by dynamic programming; since BFS is optimal under a constant branch probability of $\frac{1}{2}$, the comparison isolates the value of per-node prediction. On LCB V6 with Qwen3.5-9B, we evaluate PreSTO-PowerMH at $N_{\text{pf}} \in \{10, 20\}$ and PreSTO-EntropyCut at $N_{\text{pf}} \in \{4, 8, 12, 20\}$. From Figure 7, we find that **(1)** The predictive rule has a higher mean speedup than BFS in all six settings, from 1.51–2.11× to 1.87–2.46×; the gain is largest for PreSTO-EntropyCut at $N_{\text{pf}} \leq 12$, where cumulative model-call time falls by 17–21%. **(2)** The gain comes from more transitions per call (up to +19%), not from shorter calls: time per call is never longer than under BFS. **(3)** In the other three settings, PreSTO-PowerMH at both budgets and PreSTO-EntropyCut at $N_{\text{pf}} = 20$, the gain is small (+0.06–0.13×) and lies within one standard deviation of cumulative time, so the two rules are not distinguishable there. Since the rule changes only which requests are issued, Proposition 1 still applies and the MH transition kernel is unchanged.

Case Study 5: Distribution of ϵ-certainty. Section 3.3 shows that most decisions are ϵ-certain for Qwen3.5-9B under PreSTO-PowerMH; Appendix E.5 checks whether this holds for other base LLMs and for EntropyCut. Under PreSTO-PowerMH, 56%–87% of edges are ϵ-certain for the Qwen models and 67%–78% for Gemma4-12B-it (Figure 15); under PreSTO-EntropyCut, 60%–92% are ϵ-certain for the four Qwen models (Figure 16). Of the 24 base LLM–dataset settings in which both samplers use the same problem set, PreSTO-EntropyCut has a certain share at least as high as PreSTO-PowerMH in 18. Fewer certain decisions leave more branches in the ϵ-rounded tree, so a fixed budget realizes fewer transitions per call. The link between certainty and runtime is correlational: the histograms count all scored decisions rather than realized acceptances, so they do not by themselves isolate certainty as the cause of runtime differences.

6 Conclusion

We study how to accelerate MH-based power sampling, whose sequential accept/reject decisions force one model call per MH transition. We propose PreSTO, which prefetches proposals for a subtree of future MH states in one batched call and follows the realized path with the original acceptance rule, leaving the MH transition kernel unchanged (Proposition 1). Because the part of a prefetched tree reached by MH is sparse and nearly unary, our predictive traversal rule allocates the prefetch budget by dynamic programming. Across five base LLMs and seven datasets, PreSTO achieves median speedups of 1.71× over PowerMH and 1.88× over EntropyCut, with statistically equivalent outputs.

AI USE STATEMENT

The authors used AI to assist with proofreading, grammar, and mathematical-notation consistency checks, and the identification of potential derivation, citation, and presentation issues. All suggestions were reviewed and verified by the authors, who take full responsibility for the accuracy, integrity, and content of the manuscript.

ETHICS STATEMENT

All authors have read and commit to adhering to the ICLR Code of Ethics. This work uses only publicly available datasets and open-source models, and does not involve human subjects or human subjects data.

REPRODUCIBILITY STATEMENT

Appendix C develops the common prefetch-tree objective and request accounting, starting from Appendix C.1. Appendix C.3 gives the predictive branch model and the dynamic-programming allocation. Appendix B describes the implementation of the proposed PreSTO-PowerMH module, including the proof in Appendix B.2 that PreSTO-PowerMH preserves the PowerMH transition kernel. Appendix D gives the experimental settings, and Appendix E collects additional experimental results.

Please find our code repository at:

`https://anonymous.4open.science/r/PreSTO-ICLR2027/`

REFERENCES

Minoo Ahmadi, Seyedarmin Azizi, Erfan Baghaei Potraghloo, Mehdi Kamal, and Massoud Pedram. Chopthin-consensus power sampling: A diversity-preserving approach to LLM decoding. In *COLM 2026 Workshop on Efficient Reasoning*, 2026.

Elaine Angelino, Eddie Kohler, Amos Waterland, Margo I. Seltzer, and Ryan P. Adams. Accelerating MCMC via parallel predictive prefetching. In *Proc. of the Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 22–31, 2014.

Aleksei Arzhantsev, Otmane Sakhi, and Nicolas Chopin. Self-consistency via marginal sharpening. *arXiv preprint arXiv:2605.28142*, 2026.

Jacob Austin, Augustus Odena, Maxwell I. Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

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. *arXiv preprint arXiv:2602.10273*, 2026.

A. E. Brockwell. Parallel Markov chain Monte Carlo simulation by pre-fetching. *Journal of Computational and Graphical Statistics*, 15(1):246–261, 2006.

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. In *Proc. of the International Conference on Machine Learning (ICML)*, volume 235, pp. 5209–5235, 2024.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023.

M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in LLMs beyond the base model? In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Zhuoming Chen, Avner May, Ruslan Svirschevski, Yuhsun Huang, Max Ryabinin, Zhihao Jia, and Beidi Chen. Sequoia: Scalable and robust speculative decoding. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

Li Du, Afra Amini, Lucas Torroba Hennigen, Xinyan Velocity Yu, Holden Lee, Jason Eisner, and Ryan Cotterell. Principled gradient-based MCMC for conditional sampling of text. In *Proc. of the International Conference on Machine Learning (ICML)*, volume 235, pp. 11663–11685, 2024.

Weihua Du, Yiming Yang, and Sean Welleck. Optimizing temperature for language models with multi-sample inference. In *Proc. of the International Conference on Machine Learning (ICML)*, volume 267, 2025.

Yahya Emara, Mauricio Barba da Costa, Chi-Chih Chang, Cameron Freer, Tim Vieira, Ryan Cotterell, and Mohamed S. Abdelfattah. Faster LLM inference via sequential Monte Carlo. *arXiv preprint arXiv:2604.15672*, 2026.

Gonçalo Rui Alves Faria, Sweta Agrawal, António Farinhas, Ricardo Rei, José Guilherme Camargo de Souza, and André F. T. Martins. QUEST: quality-aware Metropolis-Hastings sampling for machine translation. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pp. 89042–89068, 2024.

Jarad Forristal, Fatemehsadat Mireshghallah, Greg Durrett, and Taylor Berg-Kirkpatrick. A block Metropolis-Hastings sampler for controllable energy-based text generation. In *Proc. of the Annual Conference on Computational Natural Language Learning (CoNLL)*, pp. 403–413, 2023.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Hong Guo, Nianhui Guo, Christoph Meinel, and Haojin Yang. Sample where you struggle: Sharpening base model reasoning via entropy-guided power sampling. *arXiv preprint arXiv:2606.09926*, 2026.

W. K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

Andre He, Daniel Fried, and Sean Welleck. Rewarding the unlikely: Lifting GRPO beyond distribution sharpening. *arXiv preprint arXiv:2506.02355*, 2025.

Jiajun He, Zongyu Guo, José Miguel Hernández-Lobato, and Yuanqi Du. Recipes for steering and scaling LLMs via sampling. *arXiv preprint arXiv:2608.26120*, 2026.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *Proc. of the International Conference on Learning Representations (ICLR)*, 2021a.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *NeurIPS Datasets and Benchmarks*, 2021b.

Jingcheng Hu, Yinmin Zhang, Qi Han, Daxin Jiang, Xiangyu Zhang, and Heung-Yeung Shum. Open-Reasoner-Zero: An open source approach to scaling up reinforcement learning on the base model. *arXiv preprint arXiv:2503.24290*, 2025.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. In *Proc. of the International Conference on Learning Representations (ICLR)*, 2025.

Xiaotong Ji, Rasul Tutunov, Matthieu Zimmer, and Haitham Bou Ammar. Scalable power sampling: Unlocking efficient, training-free reasoning for LLMs via distribution sharpening. In *Proc. of the International Conference on Machine Learning (ICML)*, 2026.

Aayush Karan and Yilun Du. Reasoning with sampling: Your base model is smarter than you think. In *Proc. of the International Conference on Learning Representations (ICLR)*, 2026.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *Proc. of the International Conference on Machine Learning (ICML)*, volume 202, pp. 19274–19286, 2023.

Marvin Li, Aayush Karan, and Sitan Chen. Blink of an eye: a simple theory for feature localization in generative models. In *Proc. of the International Conference on Machine Learning (ICML)*, 2025.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-2: faster inference of language models with dynamic draft trees. In *Proc. of the Conference on Empirical Methods for Natural Language Processing (EMNLP)*, pp. 7421–7432, 2024a.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE: speculative sampling requires rethinking feature uncertainty. In *Proc. of the International Conference on Machine Learning (ICML)*, volume 235, pp. 28935–28948, 2024b.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. Competition-level code generation with AlphaCode. *Science*, 378(6624):1092–1097, 2022.

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. In *Proc. of the International Conference on Learning Representations (ICLR)*, 2024.

Zibin Meng, Peng Xie, and Kani Chen. Depth-entropy guided sampling for training-free LLM reasoning. *arXiv preprint arXiv:2607.09693*, 2026.

Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21(6):1087–1092, 1953.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, Chunan Shi, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. SpecInfer: Accelerating large language model serving with tree-based speculative inference and verification. In *ASPLOS*, pp. 932–949, 2024.

Fatemehsadat Mireshghallah, Kartik Goyal, and Taylor Berg-Kirkpatrick. Mix and match: Learning-free controllable text generation using energy language models. In *Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 401–415, 2022.

Sarthak Mittal, Leo Gagnon, and Guillaume Lajoie. Beyond distribution sharpening: The importance of task rewards. In *Proc. of the Annual Conference on Language Modeling (COLM)*, 2026.

Radford M. Neal. Annealed importance sampling. *Statistics and Computing*, 11(2):125–139, 2001.

Tu Nguyen, Matthieu Zimmer, Rasul Tutunov, Xiaotong Ji, and Haitham Bou Ammar. The model knows, the decoder finds: Future value guided particle power sampling. In *ICML 2026 Workshop on Structured Probabilistic Inference & Generative Modeling*, 2026.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. GPQA: A graduate-level Google-proof Q&A benchmark. In *Proc. of the Annual Conference on Language Modeling (COLM)*, 2024.

Malcolm Sambridge. A parallel tempering algorithm for probabilistic sampling and optimization. *Geophysical Journal International*, 196(1):357–374, 2014.

Hannah Scheufele, Peter Blohm, and Vikas Garg. Temperature scaling in discrete sequence (language) models. In *Proc. of the International Conference on Machine Learning (ICML)*, volume 306, 2026.

Rulin Shao, Shuyue Stella Li, Rui Xin, Scott Geng, Yiping Wang, Sewoong Oh, Simon Shaolei Du, Nathan Lambert, Sewon Min, Ranjay Krishna, Yulia Tsvetkov, Hannaneh Hajishirzi, Pang Wei Koh, and Luke Zettlemoyer. Spurious rewards: Rethinking training signals in RLVR. In *Proc. of the International Conference on Machine Learning (ICML)*, 2026.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. Blockwise parallel decoding for deep autoregressive models. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 10107–10116, 2018.

Ingvar Strid. Efficient parallelisation of Metropolis–Hastings algorithms using a prefetching approach. *Computational Statistics & Data Analysis*, 54(11):2814–2835, 2010.

Chenxia Tang, Jianchun Liu, Hongli Xu, and Liusheng Huang. Top-nσ: Eliminating noise in logit space for robust token sampling of LLM. In *Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 10758–10774, 2025.

Akiyoshi Tomihari and Issei Sato. Power distribution bridges sampling, self-reward RL, and self-distillation. *arXiv preprint arXiv:2605.04542*, 2026.

Sergey Troshin, Wafaa Mohammed, Yan Meng, Christof Monz, Antske Fokkens, and Vlad Niculae. Control the temperature: Selective sampling for diverse and high-quality LLM outputs. *arXiv preprint arXiv:2510.01218*, 2025.

Heming Xia, Zhe Yang, Qingxiu Dong, Peiyi Wang, Yongqi Li, Tao Ge, Tianyu Liu, Wenjie Li, and Zhifang Sui. Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding. In *ACL (Findings)*, pp. 7655–7671, 2024.

Guifeng Ye and Shaowen Lu. Prefetching-based multiproposal Markov chain Monte Carlo algorithm. *IEEE Trans. Artif. Intell.*, 5(9):4493–4505, 2024.

Johannes Zenn and Jonas Geiping. When are likely answers right? On sequence probability and correctness in LLMs. In *ICML 2026 Workshop on Structured Probabilistic Inference & Generative Modeling*, 2026.

Stephen Zhao, Rob Brekelmans, Alireza Makhzani, and Roger Baker Grosse. Probabilistic inference in language models via twisted sequential Monte Carlo. In *Proc. of the International Conference on Machine Learning (ICML)*, volume 235, pp. 60704–60748, 2024.

Felix Zhou, Anay Mehrotra, and Quanquan C Liu. Reasoning with sampling: Cutting at decision points. *arXiv preprint arXiv:2605.30327*, 2026.

Yuqi Zhu, Jia Li, Ge Li, Yunfei Zhao, Jia Li, Zhi Jin, and Hong Mei. Hot or cold? Adaptive temperature sampling for code generation with large language models. In *Proc. of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 437–445, 2024.

CONTENTS

1 Introduction **1**

2 Preliminaries **2**

3 Methodology **3**
3.1 Motivation . . . 3
3.2 Subtree Prefetching for MH-based Power Sharpening . . . 4
3.3 Predictive Subtree Allocation for Efficient Prefetching . . . 5

4 Related Work **6**

5 Experiments **6**
5.1 Acceleration across base LLMs and datasets . . . 6
5.2 Case Studies . . . 7

6 Conclusion **9**

A Extended Related Work **15**
A.1 Sampling from the Power Distribution . . . 15
A.2 Speculative Decoding . . . 16
A.3 MCMC Prefetching . . . 16

B Implementation and Correctness of PreSTO-PowerMH **17**
B.1 Computing log-acceptance probabilities . . . 17
B.2 Exactness of Suffix Prefetching . . . 18
B.3 Examples of prefix sharing and MH transitions through the prefetched subtree . . . 19

C Prefetch Tree Selection under a Request Budget **20**
C.1 Setup and Notation . . . 20
C.2 The Transition-Count Objective . . . 21
C.3 Predictive Subtree Allocation by Dynamic Programming . . . 22

D Experimental Settings **24**
D.1 Choice of Baselines . . . 24
D.2 Choice of Datasets . . . 25
D.3 Choice of Base LLMs . . . 25
D.4 Evaluation Metrics . . . 26
D.5 Software and Hardware Configuration . . . 27
D.6 Hyperparameter Configuration . . . 27

E Additional Experimental Results **29**
E.1 Paired execution results across base LLMs . . . 29
E.2 Prefetch-budget traces . . . 29
E.3 Log-likelihood and confidence . . . 33
E.4 Predicting ϵ-certainty before generation . . . 35
E.5 Edge certainty across models and datasets . . . 38
E.6 Example prefetched-subtree trajectory . . . 40

A Extended Related Work

PreSTO draws on three lines of work. Power-sampling methods define the target and the samplers that PreSTO accelerates. Speculative decoding and MCMC prefetching are the two established ways to break serial dependence during sampling. For each line, we summarize the problem it addresses, the mechanism its methods share, and the trade-offs among them; Section 4 compares PreSTO with each line directly.

A.1 Sampling from the Power Distribution

Power sampling draws complete responses from $\pi_\alpha \propto p_0^\alpha$ with $\alpha > 1$ (Karan & Du, 2026). Raising the sequence likelihood to a power concentrates probability on responses that the base model finds likely as a whole, the same construction that annealing and parallel tempering use to control how concentrated a target is (Neal, 2001; Sambridge, 2014).

Why token-level sharpening is not enough. Token-level temperature scaling and filtering (Tang et al., 2025; Zhu et al., 2024; Du et al., 2025; Troshin et al., 2025) sharpen each next-token distribution and renormalize at every prefix. The exact next-token conditional of the power target is instead

$$\pi_\alpha(x_t \mid \mathbf{x}_{0:t-1}) \;\propto\; p_0(x_t \mid \mathbf{x}_{0:t-1})^\alpha \, Z_\alpha(\mathbf{x}_{0:t}), \qquad Z_\alpha(\mathbf{x}_{0:t}) = \sum_{\mathbf{x}_{t+1:n}} p_0(\mathbf{x}_{t+1:n} \mid \mathbf{x}_{0:t})^\alpha.$$

Token-level temperature keeps the first factor and drops Z_α. For $\alpha > 1$, Z_α is large when the continuations of a prefix concentrate on a few high-likelihood completions, so the power target prefers such prefixes, whereas low-temperature decoding cannot distinguish them. Because Z_α sums over exponentially many continuations, every power-sampling method must either avoid it or approximate it. Existing methods follow three routes.

MCMC over complete sequences. MH needs the target only up to a constant, so it avoids Z_α entirely: it compares complete sequences through the ratio $\pi_\alpha(\mathbf{y})/\pi_\alpha(\mathbf{x}) = \big(p_0(\mathbf{y})/p_0(\mathbf{x})\big)^\alpha$. The main design choice is the proposal. Earlier MCMC samplers for controlled generation with energy-based targets edit one position at a time and need many steps to make coordinated changes across a sequence (Mireshghallah et al., 2022). Regenerating a complete sequence or a suffix with an LLM instead makes large, coherent edits whose proposal probabilities come from the same forward pass (Forristal et al., 2023; Faria et al., 2024). PowerMH applies suffix regeneration to π_α with a blockwise schedule (Karan & Du, 2026), and later variants keep this structure while changing where cuts are placed or how many candidates each step considers, for example using token entropy (Zhou et al., 2026; Guo et al., 2026), or adding terms to the target (Meng et al., 2026). These samplers share three costs: each transition regenerates on average about half of the response when the cut is uniform, each stage needs many transitions, and transitions are serial because every proposal is conditioned on the current state.

Forward approximations. Forward methods generate left to right and estimate Z_α rather than drop it. Particle methods propagate a population of partial sequences from a tractable proposal, such as low-temperature decoding, reweight them by the ratio of target to proposal, and resample to discard low-weight prefixes; twisted variants learn approximations of the future term so that reweighting can happen before a sequence is complete (Zhao et al., 2024). For power targets, these samplers are consistent as the number of particles grows (Azizi et al., 2026; He et al., 2026), and in practice their quality depends on resampling rules that prevent weight degeneracy and particle collapse (Ahmadi et al., 2026). Lookahead methods instead estimate $Z_\alpha(\mathbf{x}_{0:t})$ directly by sampling a few continuations for each candidate token, and use the estimate to correct the next-token distribution (Ji et al., 2026; Nguyen et al., 2026). Both families parallelize naturally across particles or rollouts. Their bias and variance, however, depend on the number of particles or rollouts, and they commit to each generated prefix instead of revising it.

Changing or amortizing the target. A third route changes what is sampled or moves the cost into training. Sharpening the sequence likelihood can favor a single high-likelihood reasoning trace over an answer supported by many distinct traces, so marginal sharpening sharpens the distribution over final answers instead (Arzhantsev et al., 2026). Distillation-style methods fine-tune the model so that ordinary sampling approximates a powered or low-temperature sequence distribution, which removes the inference-time overhead at the price of training (Tomihari & Sato, 2026; Scheufele et al., 2026). Evidence on when sharpening helps is mixed: raising sequence likelihood through decoding need not improve correctness (Zenn & Geiping, 2026), and reward-trained models can gain in ways that sharpening does not recover (Mittal et al., 2026).

A.2 Speculative Decoding

Autoregressive decoding runs one forward pass of the target model per token, and each pass is memory-bound, so the hardware can score several tokens for nearly the cost of one. Speculative decoding exploits this. A cheap drafter proposes several future tokens, the target model scores all of them in one pass, and a verification rule commits the accepted prefix together with one token drawn from a corrected distribution (Stern et al., 2018; Leviathan et al., 2023; Chen et al., 2023; Xia et al., 2024). With the rejection rule of speculative sampling, the committed tokens follow the target model's distribution exactly, so the speedup depends only on how often drafts are accepted and on the cost of drafting.

Subsequent work follows three directions. First, drafting a tree of alternative continuations and verifying all branches in one pass with a tree-structured attention mask raises the expected number of tokens committed per pass (Miao et al., 2024). Second, better drafters, such as extra decoding heads on the target model or feature-level autoregression, raise acceptance rates (Cai et al., 2024; Li et al., 2024b). Third, because a tree of fixed size can take many shapes, tree construction becomes a budget-allocation problem: branches are added where the drafter is confident (Li et al., 2024a), or the tree shape is optimized by dynamic programming over estimated acceptance rates (Chen et al., 2024). Relaxing exact rejection gives a different trade-off. SMC-based variants weight and resample draft blocks instead of truncating at the first rejected token, accept a finite-particle approximation in exchange, and have been applied to power targets (Emara et al., 2026).

A.3 MCMC Prefetching

Parallel MCMC methods either run many independent chains, parallelize the evaluation of the target density, or speculate on the future of a single chain. Prefetching is the third approach (Brockwell, 2006; Strid, 2010). It rests on one observation: from any state, the next d accept/reject outcomes form a binary tree of depth d, and the chain follows exactly one root-to-leaf path. Proposals for every node can be drawn in advance, so the expensive target evaluations at all nodes can run in parallel. The realized path is then read off with the usual acceptance uniforms, so the chain has exactly the law of sequential MH.

Evaluating the whole tree is wasteful. It has $2^d - 1$ nodes, of which only d lie on the realized path, so the speedup obtained from p parallel workers grows only logarithmically in p (Brockwell, 2006). Later methods therefore evaluate only the nodes most likely to lie on the realized path. Static prefetching uses the acceptance rate: when acceptance is low, rejection branches dominate, and the tree can be restricted to long rejection chains (Strid, 2010). Predictive prefetching uses a cheap approximation of the target to estimate each branch probability, and still decides every realized transition with the exact target (Angelino et al., 2014).

A different approach parallelizes within one transition. Multiproposal MH draws several candidates from the current state and moves according to a Markov chain over the candidate set, which preserves the stationary distribution but changes the transition kernel. This can be combined with prefetching by expanding a tree of multiproposal steps (Ye & Lu, 2024).

B Implementation and Correctness of PreSTO-PowerMH

Algorithms 1 and 2 place the original PowerMH sampler and PreSTO-PowerMH side by side. Both share the same blockwise outer loop and the same acceptance rule; they differ only in how the K MH steps of a stage are executed. PowerMH issues one proposal-LLM call per MH step and must wait for each accept/reject outcome before forming the next proposal. PreSTO-PowerMH issues one *batched* call per prefetch round and realizes $k_\Delta \geq 1$ MH steps from it. In Algorithm 2, PrefetchAndScore builds the prefetching tree rooted at the current state $\mathbf{x}$, collects at most N_{pf} feasible requests under the tree traversal rule, generates and scores them in one batched call, and returns the scored prefetched subtree $\mathcal{T}$; deferred requests keep their cut indices across rounds. MHTransition then follows the realized accept/reject path through $\mathcal{T}$ and returns the reached state $\mathbf{x}_\Delta$ together with the number k_Δ of transitions completed.

Algorithm 1 PowerMH (Karan & Du, 2026)

Common settings: the base-model distribution p_0, proposal-model distribution q_τ, power-sharpening coefficient α, response length n, block size B, and MCMC steps per block K.
Output: the final state $\mathbf{x} = \mathbf{x}_{0:n}$ after K MH steps per block, approximately distributed as π_α.

for $\ell \leftarrow 1, \ldots, n/B$ **do**
  Extend $\mathbf{x}_{0:L}$ to $\mathbf{x}_{0:T}$ using q_τ.
  for $k \leftarrow 1, \ldots, K$ **do**
    $\mathbf{y} \leftarrow$ DrawOneCandidate$(\mathbf{x})$.
    $A \leftarrow$ ComputeAcceptance$(\mathbf{y}, \mathbf{x})$.
    $u \sim \text{Unif}(0, 1)$;
    if $u \leq A$, set $\mathbf{x} \leftarrow \mathbf{y}$.
return $\mathbf{x}$.

Algorithm 2 PreSTO-PowerMH (ours)

PreSTO-PowerMH only settings: tree traversal rule, prefetch budget N_{pf}.
Output: the final state $\mathbf{x} = \mathbf{x}_{0:n}$ after K MH steps per block, approximately distributed as π_α.

for $\ell \leftarrow 1, \ldots, n/B$ **do**
  Extend $\mathbf{x}_{0:L}$ to $\mathbf{x}_{0:T}$ with q_τ.
  $k \leftarrow 0$.
  while $k < K$ **do**
    $\mathcal{T} \leftarrow$ PrefetchAndScore$(\mathbf{x}, N_{\text{pf}})$.
    $(\mathbf{x}_\Delta, k_\Delta) \leftarrow$ MHTransition$(\mathcal{T}, \mathbf{x})$.
    update MH chain $k \leftarrow k + k_\Delta$; $\mathbf{x} \leftarrow \mathbf{x}_\Delta$.
return $\mathbf{x}$.

B.1 Computing log-acceptance probabilities

We follow the standard steps to derive a simplified expression for the suffix-resampling log-acceptance probability in equation 5 by specializing the generic MH acceptance rule in equation 1. All states share the same length T: every proposal is generated to the full horizon and does not stop at the end-of-sequence token. Fix a current state $\mathbf{x} = \mathbf{x}_{0:T}$ and a cut index c. At this cut, the proposal retains the prefix $\mathbf{x}_{0:c}$ and samples a new suffix $\mathbf{y}_{c+1:T} \sim q_\tau(\cdot \mid \mathbf{x}_{0:c})$ from the proposal LLM. The resulting candidate is $\mathbf{y} = (\mathbf{x}_{0:c}, \mathbf{y}_{c+1:T})$, so $\mathbf{y}_{0:c} = \mathbf{x}_{0:c}$. We write $q(\mathbf{y} \mid \mathbf{x}, c)$ for the MH proposal kernel conditional on the cut, to distinguish it from the token-level LLM conditionals q_τ. PowerMH draws $c \sim \text{Unif}\{0, \ldots, T-1\}$ independently of the state, and the reverse move from $\mathbf{y}$ to $\mathbf{x}$ uses the same cut. The factor $1/T$ therefore appears in both the forward and the reverse proposal probabilities and cancels. Each cut-conditional kernel with the acceptance probability below is π_α-reversible, and so is their uniform mixture over c. For this candidate, the acceptance rule in equation 1 becomes

$$A(\mathbf{y}, \mathbf{x}) = \min\left\{1,\ \frac{\pi_\alpha(\mathbf{y}_{1:T} \mid \mathbf{x}_0)\, q(\mathbf{x} \mid \mathbf{y}, c)}{\pi_\alpha(\mathbf{x}_{1:T} \mid \mathbf{x}_0)\, q(\mathbf{y} \mid \mathbf{x}, c)}\right\}. \tag{6}$$

We next simplify the proposal and target ratios in equation 6 separately.

1. **Proposal log-ratio.** At the fixed cut c, the candidate differs from $\mathbf{x}$ only in the resampled suffix, so the forward proposal probability is

$$q(\mathbf{y} \mid \mathbf{x}, c) = q_\tau\big(\mathbf{y}_{c+1:T} \mid \mathbf{x}_{0:c}\big).$$

The reverse move applies the same cut c to propose $\mathbf{x}$ from $\mathbf{y}$. It preserves $\mathbf{y}_{0:c}$ and generates the original suffix $\mathbf{x}_{c+1:T}$. Using $\mathbf{y}_{0:c} = \mathbf{x}_{0:c}$, the reverse proposal probability becomes

$$q(\mathbf{x} \mid \mathbf{y}, c) = q_\tau\big(\mathbf{x}_{c+1:T} \mid \mathbf{y}_{0:c}\big) = q_\tau\big(\mathbf{x}_{c+1:T} \mid \mathbf{x}_{0:c}\big).$$

Therefore, the log-proposal ratio is

$$\log \frac{q(\mathbf{x} \mid \mathbf{y}, c)}{q(\mathbf{y} \mid \mathbf{x}, c)} = \log q_\tau\big(\mathbf{x}_{c+1:T} \mid \mathbf{x}_{0:c}\big) - \log q_\tau\big(\mathbf{y}_{c+1:T} \mid \mathbf{x}_{0:c}\big). \tag{7}$$

2. **Target log-ratio.** By equation 2, $\pi_\alpha(\cdot \mid \mathbf{x}_0) \propto p_0(\cdot \mid \mathbf{x}_0)^\alpha$. Because the numerator and denominator use the same prompt and the same fixed length T, their common normalizing constant cancels:
$$\frac{\pi_\alpha(\mathbf{y}_{1:T} \mid \mathbf{x}_0)}{\pi_\alpha(\mathbf{x}_{1:T} \mid \mathbf{x}_0)} = \left(\frac{p_0(\mathbf{y}_{1:T} \mid \mathbf{x}_0)}{p_0(\mathbf{x}_{1:T} \mid \mathbf{x}_0)}\right)^\alpha.$$
Since $\mathbf{y}_{1:c} = \mathbf{x}_{1:c}$, autoregressively factorizing each term on the right-hand side at the cut gives
$$p_0(\mathbf{y}_{1:T} \mid \mathbf{x}_0) = p_0(\mathbf{x}_{1:c} \mid \mathbf{x}_0)\, p_0(\mathbf{y}_{c+1:T} \mid \mathbf{x}_{0:c}),$$
$$p_0(\mathbf{x}_{1:T} \mid \mathbf{x}_0) = p_0(\mathbf{x}_{1:c} \mid \mathbf{x}_0)\, p_0(\mathbf{x}_{c+1:T} \mid \mathbf{x}_{0:c}).$$
Cancelling the common prefix probability $p_0(\mathbf{x}_{1:c} \mid \mathbf{x}_0)$ and taking logarithms yields
$$\log \frac{\pi_\alpha(\mathbf{y}_{1:T} \mid \mathbf{x}_0)}{\pi_\alpha(\mathbf{x}_{1:T} \mid \mathbf{x}_0)} = \alpha\Big(\log p_0\big(\mathbf{y}_{c+1:T} \mid \mathbf{x}_{0:c}\big) - \log p_0\big(\mathbf{x}_{c+1:T} \mid \mathbf{x}_{0:c}\big)\Big). \tag{8}$$

3. **Combined log-ratio.** Taking the logarithm of the acceptance rule in equation 6, using $\log \min\{1, r\} = \min\{0, \log r\}$, and substituting equation 7 and equation 8 yields
$$\begin{aligned}\log A(\mathbf{y}, \mathbf{x}) = \min\Big\{0,\ &\log q_\tau(\mathbf{x}_{c+1:T} \mid \mathbf{x}_{0:c}) - \log q_\tau(\mathbf{y}_{c+1:T} \mid \mathbf{x}_{0:c}) \\ &+ \alpha\big(\log p_0(\mathbf{y}_{c+1:T} \mid \mathbf{x}_{0:c}) - \log p_0(\mathbf{x}_{c+1:T} \mid \mathbf{x}_{0:c})\big)\Big\},\end{aligned} \tag{9}$$
which is exactly equation 5. Each suffix log-probability on the right-hand side is evaluated by summing token-level conditional log-probabilities from the serving engine. For example,
$$\log q_\tau(\mathbf{y}_{c+1:T} \mid \mathbf{x}_{0:c}) = \sum_{t=c+1}^{T} \log q_\tau(y_t \mid \mathbf{x}_{0:c}, \mathbf{y}_{c+1:t-1}).$$
Evaluating these quantities in the log domain avoids underflow from multiplying many small token probabilities and improves numerical stability.

In implementation, both distributions use the same LLM. A single batched model call generates candidate suffixes and provides their log-probabilities under both q_τ and p_0, using the same logits with and without temperature scaling. For proposal temperature $\tau > 0$ and vocabulary $\mathcal{V}$, this gives
$$q_\tau(y_t \mid \mathbf{y}_{0:t-1}) = \frac{p_0(y_t \mid \mathbf{y}_{0:t-1})^{1/\tau}}{\sum_{v \in \mathcal{V}} p_0(v \mid \mathbf{y}_{0:t-1})^{1/\tau}}. \tag{10}$$

B.2 Exactness of Suffix Prefetching

Reproducing the sequential chain under speculative evaluation is a standard property of MCMC prefetching (Brockwell, 2006; Angelino et al., 2014). The following result specializes this argument to our suffix-resampling scheme. We include the proof for completeness, verifying that generating proposals early under Condition 1 does not change the sequential PowerMH trajectory. Appendix C then analyzes how many transitions a prefetched subtree yields and which requests to prefetch under a request budget.

Proposition 1. *Fix an integer $K \geq 0$. For $k = 0, \ldots, K$, let $\mathbf{x}_{\mathrm{pf}}^{(k)}$ and $\mathbf{x}_{\mathrm{MH}}^{(k)}$ denote the states after k committed transitions of PreSTO-PowerMH and PowerMH, respectively. Suppose both samplers start from the same state and PreSTO-PowerMH satisfies the following assumptions:*

(A1) Prefetched proposals satisfy Condition 1 and are generated from the prefix required by the corresponding decision.

(A2) Each decision is assigned independent cut, suffix-sampling, and acceptance random inputs with the PowerMH laws, independently across decisions. Scheduling and deferral preserve these inputs, and every proposal required by the realized trajectory is eventually evaluated.

(A3) Every committed transition uses its assigned proposal and the exact MH acceptance rule equation 1. Both children are retained as possible outcomes; reaching a leaf postpones subsequent decisions to a later batch.

Then a coupling that uses the same random inputs at corresponding decisions preserves both sampling laws and satisfies
$$\mathbf{x}_{\mathrm{pf}}^{(k)} = \mathbf{x}_{\mathrm{MH}}^{(k)}, \qquad \text{for every } k = 0, \ldots, K.$$
Consequently, if P_{MH} denotes the PowerMH transition kernel, the kernel of PreSTO-PowerMH after K committed transitions is P_{MH}^K.

Proof. Assign independent PowerMH random inputs to each decision of the full accept–reject tree, and couple the two executions through these inputs. Whether a decision is reached in the eventual committed trajectory depends only on its ancestors' inputs, so its own inputs retain their original conditional law. Inclusion in the current batch may depend on scheduling; this changes only when the proposal is evaluated. Preserving deferred inputs and eventually evaluating every required proposal makes the coupling valid across batch boundaries.

Suppose the executions agree after $k < K$ committed transitions. By Condition 1 and the prefix identity established in Section 3.2, any intervening accepted proposal preserves the prefix required by a feasible prefetched request. Rejections leave the state unchanged. A deferred request is evaluated from the required prefix when it becomes available. Thus, both executions use the same prefix in the suffix proposal defined by equation 4, and their common suffix-sampling randomness produces the same candidate.

The current states and candidates therefore coincide, giving the same MH acceptance probability in equation 1. The shared acceptance uniform yields the same decision and next state. Since the initial states agree, induction establishes equality through transition K. Each committed transition consequently has kernel P_{MH}, and the K-step kernel is P_{MH}^{K}. □

Remark. The proof uses PowerMH only through the laws of the per-decision random inputs in (A2) and the acceptance rule in (A3). EntropyCut (Zhou et al., 2026) keeps the suffix-resampling proposal in equation 4 and changes only the law of the cut index, which depends on the base model's next-token entropies along the current state, together with the MH correction for this cut law. Each node of the prefetching tree fixes the state at its decision, so PreSTO-EntropyCut can draw that decision's cut from the EntropyCut law and evaluate the EntropyCut acceptance rule. Replacing the PowerMH laws in (A2) and the acceptance rule in (A3) with those of EntropyCut, the same coupling argument gives $\mathbf{x}_{\mathrm{pf}}^{(k)} = \mathbf{x}_{\mathrm{EC}}^{(k)}$ for every $k = 0, \ldots, K$, where $\mathbf{x}_{\mathrm{EC}}^{(k)}$ is the state after k transitions of EntropyCut. Hence PreSTO-EntropyCut realizes the K-step EntropyCut kernel P_{EC}^{K} and, like PreSTO-PowerMH, changes only when proposals are generated, not the sampling law.

B.3 Examples of prefix sharing and MH transitions through the prefetched subtree

Example 4. Suppose $\mathbf{x}_{0:T}$ is cached and $c_2 \le c_1$. In Figure 8(a), the requests for $\mathbf{y}$ and $\mathbf{z}$ condition on $\mathbf{x}_{0:c_1}$ and $\mathbf{x}_{0:c_2}$, respectively. Both requests reuse the cached prefix $\mathbf{x}_{0:c_2}$; the request for $\mathbf{y}$ also reuses $\mathbf{x}_{c_2+1:c_1}$. Figure 8(c) represents these shared KV-cache segments as a radix tree, as in `SGLang`. `vLLM` instead reuses full, fixed-size KV-cache blocks, indexed by hashes that incorporate each block's token IDs and the preceding block's hash. Both implementations avoid repeated computation of cached prefix states while preserving the proposal distribution and autoregressive decoding of each requested suffix.

Example 5. After generating the resampled sequences in Example 3, we associate each prefetched proposal with its MH log-acceptance probability. In Figure 8(a), request (v_1, c_1) produces $\mathbf{y}$ by resampling the suffix of the cached state $\mathbf{x}$ after cut index c_1. Using equation 5 with cut index c_1, we compute $\log A(\mathbf{y}, \mathbf{x})$ and associate it with the acceptance edge $v_0 \to v_1$ in Figure 8(b). Likewise, request (v_3, c_2) proposes $\mathbf{z}$ from $\mathbf{y}$. We compute $\log A(\mathbf{z}, \mathbf{y})$ using cut index c_2 and associate it with the acceptance edge $v_1 \to v_3$.

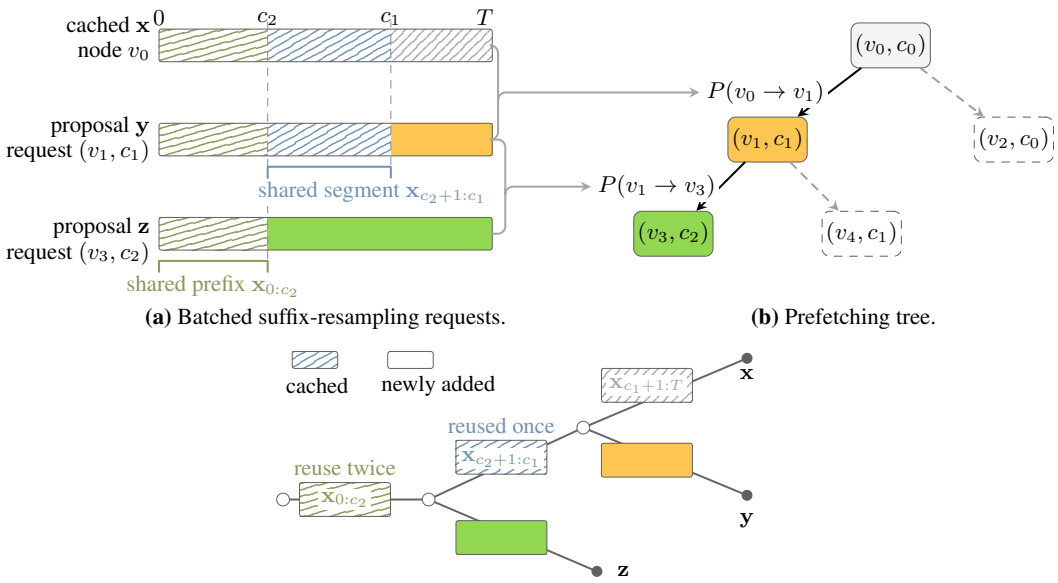
We simulate the sequential MH decisions by traversing the prefetched subtree in Figure 8(b) from its root. At node v_0, draw $u_1 \sim \mathrm{Unif}(0, 1)$ and select the next node as

$$v_{\mathrm{next}} = \begin{cases} v_1, & \text{if } u_1 \le P(v_0 \to v_1), \\ v_2, & \text{otherwise,} \end{cases} \tag{11}$$

where $P(v_0 \to v_1) = A(\mathbf{y}, \mathbf{x})$ is the MH acceptance probability. The realization in Figure 2 accepts $\mathbf{y}$, so the traversal proceeds from v_0 to v_1. At v_1, draw $u_2 \sim \mathrm{Unif}(0, 1)$ independently. In this realization, $u_2 > A(\mathbf{z}, \mathbf{y})$, so the second proposal $\mathbf{z}$ is rejected, yielding the tree path $v_0 \to v_1 \to v_4$ and the corresponding MH trajectory

$$\mathbf{x}^{(0)} = \mathbf{x} \to \mathbf{x}^{(1)} = \mathbf{y} \to \mathbf{x}^{(2)} = \mathbf{y}.$$

Together with Examples 1, 2, and 3, this example shows how PreSTO-PowerMH batches two suffix-resampling requests into one proposal-LLM call while preserving sequential MH decisions. Along the illustrated path, this batched call realizes two MH transitions that require two serial proposal-LLM calls in PowerMH.



(a) Batched suffix-resampling requests. **(b)** Prefetching tree.

(c) Radix-tree layout of the KV cache.

Figure 8: Two batched suffix-resampling requests, their prefetching tree, and their shared KV cache. **(a)** Both requests reuse the cached prefix $\mathbf{x}_{0:c_2}$, and the request for **y** also reuses $\mathbf{x}_{c_2+1:c_1}$. **(b)** Each acceptance (left) edge carries the MH acceptance probability computed from the pair of sequences it connects in (a). **(c)** Radix-tree representation of the shared KV-cache segments.

C Prefetch Tree Selection under a Request Budget

Appendix C.1 fixes notation for the prefetching tree. Appendix C.2 derives the expected transition count of a prefetched subtree and its request cost. Appendix C.3 then poses the budgeted allocation objective and solves it by dynamic programming.

C.1 Setup and Notation

We use the prefetching tree of Section 3.2, rooted at the node v_0 that holds the current MH state. Expanding a node u creates an acceptance child u^+ and a rejection child u^-. Every node v carries a cut index $c(v)$, assigned before any proposal is generated or scored: the root has $c(v_0) = T$, an acceptance child u^+ carries the cut index drawn for its suffix resampling, and a rejection child inherits $c(u^-) = c(u)$ because it retains the state of u. The depth $d(v)$ of a node is the number of edges on the path from v_0 to v. We cap the tree at a depth limit D and write $\mathcal{T}_D$ for the resulting finite tree. The cap is needed because the chain of rejection children from the root always satisfies Condition 1, so the feasible tree would otherwise be infinite. We use the following terms throughout this appendix.

- *Expandable node.* A node $u \in \mathcal{T}_D$ with $d(u) < D$ is expandable if the request at its acceptance child is feasible, that is, $c(u^+) \le c(u)$ as in Condition 1.
- *Decision.* A decision is an expandable node whose strict ancestors are all expandable, and $\mathcal{I}_{\max}$ denotes the set of decisions. Once the proposal at u^+ is generated, the traversal moves from a decision u to u^+ if the proposal is accepted and to u^- if it is rejected.
- *Ancestor-closed.* A set $\mathcal{I} \subseteq \mathcal{I}_{\max}$ is ancestor-closed if, for every $u \in \mathcal{I}$, every strict ancestor of u also lies in $\mathcal{I}$.
- *Selection.* A selection is any ancestor-closed $\mathcal{I} \subseteq \mathcal{I}_{\max}$; its elements are the expanded decisions.
- *Prefetched subtree.* Expanding every $u \in \mathcal{I}$ gives the root v_0 together with both children of every $u \in \mathcal{I}$. Its internal nodes are exactly the decisions in $\mathcal{I}$, and its leaves are the children outside $\mathcal{I}$.

Any $D \ge N_{\mathrm{pf}}$ leaves the optimum of equation 20 unchanged, because an ancestor-closed set of at most N_{pf} decisions contains no decision deeper than $N_{\mathrm{pf}} - 1$.

Definition 1. *Fix a realization of the proposals and of their acceptance probabilities, with ϵ as in Section 3.3.*
Rounded branch probabilities. *At each decision u,*

$$h(u^+;\epsilon) := \begin{cases} 0, & A(u^+,u) \le \epsilon, \\ 1, & A(u^+,u) \ge 1-\epsilon, \\ A(u^+,u), & \textit{otherwise}, \end{cases} \qquad h(u^-;\epsilon) := 1 - h(u^+;\epsilon). \tag{12}$$

Traversal. *Fix a selection $\mathcal{I}$, so that the decisions in $\mathcal{I}$ are the expanded ones. A traversal started at u follows one child of each decision in $\mathcal{I}$ that it reaches, moving from a reached decision w to w^+ with probability $h(w^+;\epsilon)$ and to w^- with probability $h(w^-;\epsilon)$, and stops at the first node outside $\mathcal{I}$. The choices at different decisions are independent. Write $F_{\mathcal{I}}(u;\epsilon)$ for the expected number of transitions completed by a traversal started at u.*
Reaching weights. *The reaching weights accumulate these branch probabilities along each root-to-node path:*

$$R(v_0;\epsilon) = 1, \qquad R(v;\epsilon) = R(u;\epsilon)h(v;\epsilon) \quad \textit{along each edge } (u,v). \tag{13}$$

C.2 The Transition-Count Objective

To compare feasible sets of requests by the number of MH transitions they complete, this subsection derives the expected transition count at the root, following prior prefetching analyses (Strid, 2010), and counts the requests that each selection issues.
Fix a selection $\mathcal{I}$ and let $F_{\mathcal{I}}(u;\epsilon)$ be the expected transition count of Definition 1. Each decision in $\mathcal{I}$ that the traversal visits completes one transition, whether its proposal is accepted or rejected, and $F_{\mathcal{I}}(u;\epsilon) = 0$ when $u \notin \mathcal{I}$. Lemma 1 shows that the root count $F_{\mathcal{I}}(v_0;\epsilon)$ equals the sum of reaching weights over $\mathcal{I}$.

Lemma 1 (Expected transition count)**.** *With reaching weights $R(u;\epsilon)$ defined by equation 13, the expected transition count from the root satisfies*

$$F_{\mathcal{I}}(v_0;\epsilon) = \sum_{u\in\mathcal{I}} R(u;\epsilon). \tag{14}$$

Proof. A traversal starting at v_0 visits each node at most once. Each visited decision in $\mathcal{I}$ completes exactly one transition. Thus the transition count equals the number of nodes in $\mathcal{I}$ visited by the traversal. By linearity of expectation,

$$F_{\mathcal{I}}(v_0;\epsilon) = \sum_{u\in\mathcal{I}} \Pr(\text{the traversal visits } u).$$

For any $u \in \mathcal{I}$, ancestor closure ensures that every strict ancestor of u is expanded. Hence no unexpanded node can stop the traversal along the unique path from v_0 to u. The traversal therefore visits u exactly when it follows the branches along that path. The probability of this event is the product of the corresponding branch probabilities, which equals $R(u;\epsilon)$ by equation 13. Consequently,

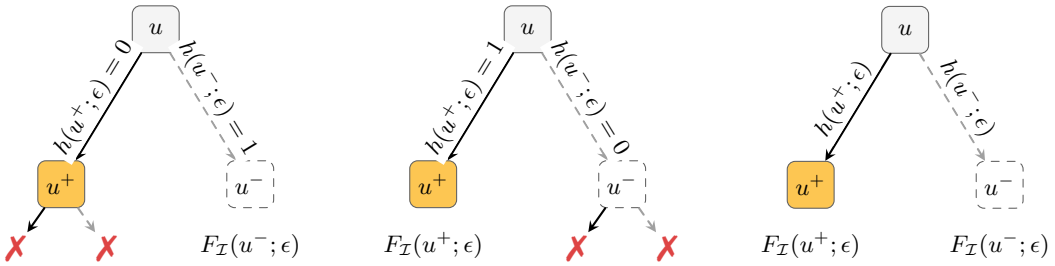
$$F_{\mathcal{I}}(v_0;\epsilon) = \sum_{u\in\mathcal{I}} R(u;\epsilon).$$

When $\mathcal{I} = \varnothing$, both the transition count and the empty sum are zero, so the identity still holds. □

Remark. $F_{\mathcal{I}}(\cdot;\epsilon)$ can also be computed recursively. If u is unexpanded, the traversal stops on arrival, so $F_{\mathcal{I}}(u;\epsilon) = 0$. If instead $u \in \mathcal{I}$, a traversal started at u completes that decision's transition and then continues into whichever child the outcome selects. We have

$$F_{\mathcal{I}}(u;\epsilon) = 1 + h(u^+;\epsilon)F_{\mathcal{I}}(u^+;\epsilon) + h(u^-;\epsilon)F_{\mathcal{I}}(u^-;\epsilon), \qquad u \in \mathcal{I}. \tag{15}$$

Each child value is again conditional on starting at that child, so no root-to-node reaching weight enters the recurrence. Figure 9 provides a visualization of equation 15. Certain rejection and certain acceptance correspond to $h(u^+;\epsilon) = 0$ and $h(u^+;\epsilon) = 1$, respectively. Expansions below a zero-weight child can be omitted without changing $F_{\mathcal{I}}(u;\epsilon)$, while the leading 1 still counts that decision's own transition.



(a) ϵ-certain rejection **(b)** ϵ-certain acceptance **(c)** ϵ-uncertain decision

Figure 9: Visualization of equation 15: (a) certain rejection, (b) certain acceptance, and (c) an uncertain rounded decision. A ✗ marks a pruned edge. The proposal at u^+ is required in every case.

Request accounting. The request count depends only on the selected decisions $\mathcal{I}$, not on the traversal. Expanding a decision u issues exactly one request, for the proposal at its acceptance child u^+; the rejection child u^- retains the state of u and needs no proposal. Expanding the decisions in $\mathcal{I}$ therefore issues $|\mathcal{I}|$ requests and yields a prefetched subtree of $2|\mathcal{I}|+1$ nodes. A request is charged when it is issued, not when it is used: the traversal may never reach its decision, and when it does, the proposal is charged whether it is accepted or rejected. Rounding creates no free expansions: at a certain rejection, $h(u^+;\epsilon)=0$ makes the acceptance branch unreachable, yet the proposal at u^+ is exactly the request that expanding u pays for.

C.3 Predictive Subtree Allocation by Dynamic Programming

The ϵ-certainty categorization in equation 12 depends on the acceptance probability $A(u^+,u)$, which is known only once the proposal at u^+ has been generated and scored. It is therefore unavailable while we assemble the request batch, so we instead use a simple neural network model to predict how likely that acceptance probability is to fall near zero or near one. Throughout this subsection, $A := A(u^+,u)$ denotes the acceptance probability of the not-yet-generated proposal at u^+, viewed as a random variable, and conditioning on u means conditioning on the information available before allocation, such as the token-wise likelihood and entropy from the proposal LLM.

Suppose that for a decision u we have a predicted probability $\widehat{P}(A \le \epsilon \mid u)$ that it is an ϵ-certain reject and a predicted probability $\widehat{P}(A \ge 1-\epsilon \mid u)$ that it is an ϵ-certain accept. We then set the *predicted branch probabilities* to

$$\begin{aligned}\widehat{h}(u^+;\epsilon) &= \tfrac{1}{2}\big[1+\widehat{P}(A\ge 1-\epsilon \mid u)-\widehat{P}(A\le\epsilon\mid u)\big],\\ \widehat{h}(u^-;\epsilon) &= 1-\widehat{h}(u^+;\epsilon).\end{aligned} \tag{16}$$

Lemma 2 justifies this choice.

Lemma 2 (Midpoint branch probability)**.** *Fix a decision u. Its exact certainty-class probabilities bound the mean rounded branch probability by*

$$\begin{aligned}\mathbb{E}[h(u^+;\epsilon)\mid u] &\ge P(A\ge 1-\epsilon\mid u)+\epsilon\,P(\epsilon<A<1-\epsilon\mid u),\\ \mathbb{E}[h(u^+;\epsilon)\mid u] &\le P(A\ge 1-\epsilon\mid u)+(1-\epsilon)\,P(\epsilon<A<1-\epsilon\mid u).\end{aligned} \tag{17}$$

The error *of a candidate value $\widehat{h}(u^+;\epsilon)$ is $\big|\widehat{h}(u^+;\epsilon)-\mathbb{E}[h(u^+;\epsilon)\mid u]\big|$, and its* worst-case error *is the largest such error over the means allowed by equation 17. The worst-case error is smallest at the midpoint*

$$\widehat{h}(u^+;\epsilon)=\tfrac{1}{2}\big[1+P(A\ge 1-\epsilon\mid u)-P(A\le\epsilon\mid u)\big]. \tag{18}$$

Proof. By equation 12, $h(u^+;\epsilon)$ equals one on the certain-accept class, zero on the certain-reject class, and A on the uncertain class, so

$$\mathbb{E}[h(u^+;\epsilon)\mid u]=P(A\ge 1-\epsilon\mid u)+\mathbb{E}\big[A\mathbf{1}\{\epsilon<A<1-\epsilon\}\;\big|\;u\big].$$

The terms on the right-hand side are:

- $P(A\ge 1-\epsilon\mid u)$ from the certain-accept class, whose factor is one;

- $\mathbb{E}[A\mathbf{1}\{\epsilon < A < 1-\epsilon\} \mid u]$ from the uncertain class, whose factor is A itself.

The certain-reject class has factor zero and so contributes no term.
On the uncertain class $\epsilon < A < 1-\epsilon$, so the remaining expectation lies between $\epsilon\, P(\epsilon < A < 1-\epsilon \mid u)$ and $(1-\epsilon)\, P(\epsilon < A < 1-\epsilon \mid u)$, which gives equation 17. If a quantity is known only to lie in a given interval, the estimate whose worst-case error is smallest is that interval's midpoint, here

$$\widehat{h}(u^+;\epsilon) = P(A \geq 1-\epsilon \mid u) + \tfrac{1}{2} P(\epsilon < A < 1-\epsilon \mid u).$$

Substituting $P(\epsilon < A < 1-\epsilon \mid u) = 1 - P(A \geq 1-\epsilon \mid u) - P(A \leq \epsilon \mid u)$ gives the final result equation 18. □

Lemma 2 uses the exact class probabilities P, which are unknown before the proposal at u^+ is scored. Evaluating the midpoint equation 18 at the predicted class probabilities $\widehat{P}$ instead gives equation 16, so the two differ only through the prediction error in $\widehat{P}$.

Objective formulation. Once $\widehat{h}$ is available at every decision, the predicted reaching weights $\widehat{R}$ follow the recursion of equation 13 with $\widehat{h}$ in place of h:

$$\widehat{R}(v_0;\epsilon) = 1, \qquad \widehat{R}(v;\epsilon) = \widehat{R}(u;\epsilon)\widehat{h}(v;\epsilon) \quad \text{along each edge } (u,v). \tag{19}$$

For a selection $\mathcal{I}$ and a node u, let $\widehat{F}_{\mathcal{I}}(u;\epsilon)$ denote the expected number of transitions completed by a traversal started at u when the branch probabilities h in Definition 1 are replaced by $\widehat{h}$. Lemma 1 and the recursion equation 15 carry over with $\widehat{h}$ in place of h; in particular, $\widehat{F}_{\mathcal{I}}(v_0;\epsilon) = \sum_{u\in\mathcal{I}} \widehat{R}(u;\epsilon)$. Given a predictive model, our objective (based on Lemma 1) is to choose the selection $\mathcal{I}$ that maximizes the predicted transition count under a request budget. By the request accounting in Appendix C.2, a selection $\mathcal{I}$ costs exactly $|\mathcal{I}|$ requests, so the objective is

$$\begin{aligned} \max_{\mathcal{I}} \quad & \widehat{F}_{\mathcal{I}}(v_0;\epsilon) = \sum_{u\in\mathcal{I}} \widehat{R}(u;\epsilon) \\ \text{s.t.} \quad & \mathcal{I} \subseteq \mathcal{I}_{\max} \text{ ancestor-closed}, \\ & |\mathcal{I}| \leq N_{\mathrm{pf}}. \end{aligned} \tag{20}$$

The candidate set $\mathcal{I}_{\max}$, determined by Condition 1 and the depth limit D, ensures that every selected request can be generated in a single batch. Cut indices never increase along a root-to-node path, so by induction the state of every decision u agrees with the root state on positions $0,\ldots,c(u)$: an accepted edge keeps the prefix up to $c(u^+) \leq c(u)$, and a rejected edge keeps the whole state. Each selected request therefore conditions only on a prefix of the root state, which is available before the batch. Ancestor closure requires every ancestor of a selected decision to be selected as well. A decision with $\widehat{R}(u;\epsilon) = 0$ contributes nothing to the objective while still costing a request, so it and its descendants may be left unexpanded without changing the optimum.

C.3.1 Optimal budget allocation via dynamic programming

Define $\widehat{V}(u,n;\epsilon)$ as the largest predicted transition count attainable from u using at most n requests,

$$\widehat{V}(u,n;\epsilon) = \max_{\substack{\mathcal{I}_u \text{ feasible below } u \\ |\mathcal{I}_u| \leq n}} \widehat{F}_{\mathcal{I}_u}(u;\epsilon), \tag{21}$$

where a set of decisions $\mathcal{I}_u$ is *feasible below* u if $\mathcal{I}_u \subseteq \mathcal{I}_{\max}$, every element of $\mathcal{I}_u$ lies in the subtree rooted at u, and every decision on the path from u to an element of $\mathcal{I}_u$ also lies in $\mathcal{I}_u$.
Optimizing equation 21 decomposes the budget at each decision: one request is reserved to expand u itself, and the remaining $n-1$ are split between its two child subtrees before the outcome is known, as illustrated in Figure 10. This decomposition is inspired by the tree-based speculative decoding of Sequoia (Chen et al., 2024).

Proposition 2 (Optimal predictive allocation)**.** *Fix ϵ and the predicted branch probabilities $\widehat{h}(u^+;\epsilon)$ and $\widehat{h}(u^-;\epsilon)$ in equation 16. For every node u and every budget $n \geq 0$,*

$$\widehat{V}(u,n;\epsilon) = \begin{cases} 0, & \text{if } u \notin \mathcal{I}_{\max}, \\ 0, & \text{if } n = 0, \\ 1 + \max\limits_{0\leq m\leq n-1} \big\{ \widehat{h}(u^+;\epsilon)\widehat{V}(u^+,m;\epsilon) + \widehat{h}(u^-;\epsilon)\widehat{V}(u^-,n-1-m;\epsilon) \big\}, & \text{else}. \end{cases} \tag{22}$$

The maximum in equation 20 equals $\widehat{V}(v_0, N_{\mathrm{pf}};\epsilon)$.

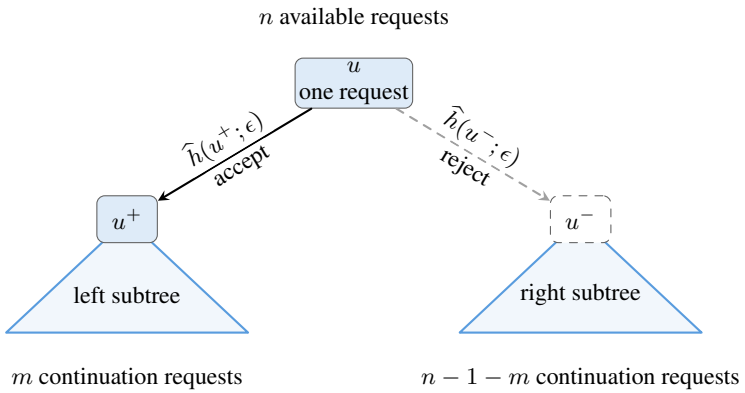


Figure 10: Budget decomposition visualization for Proposition 2.

Proof. Boundary cases. A nonempty selection feasible below u contains u, so it needs both $u \in \mathcal{I}_{\max}$ and $n \geq 1$. Either boundary case therefore leaves only the empty selection, giving $\widehat{V}(u, n; \epsilon) = 0$. Fix $u \in \mathcal{I}_{\max}$ and $n \geq 1$ below.

Upper bound. Expanding u alone has value one by equation 15, so an optimal $\mathcal{I}_u$ is nonempty and hence contains u, spending one request. Its remaining decisions split between the two child subtrees, and its restriction to each is feasible below that child. With m of them below u^+ and at most $n-1-m$ below u^-, where $0 \leq m \leq n-1$, the predictive form of equation 15 and the optimality of each child value give

$$\widehat{F}_{\mathcal{I}_u}(u;\epsilon) \leq 1 + \widehat{h}(u^+;\epsilon)\widehat{V}(u^+, m;\epsilon) + \widehat{h}(u^-;\epsilon)\widehat{V}(u^-, n-1-m;\epsilon).$$

Maximizing over m gives the upper bound in equation 22.

Attainability. Conversely, combine a maximizing m with optimal selections below u^+ and u^- at capacities m and $n-1-m$. Together with the expansion of u they are feasible below u and use at most n requests, and equation 15 gives them exactly the maximized value. The bound is attained, which proves equation 22; induction from the leaves of the finite tree $\mathcal{T}_D$ then determines every $\widehat{V}(u, n; \epsilon)$.

Feasibility below v_0 is ancestor-closedness within $\mathcal{I}_{\max}$, so $\widehat{V}(v_0, N_{\mathrm{pf}}; \epsilon)$ is the optimum of equation 20. □

D Experimental Settings

D.1 Choice of Baselines

MH-based samplers. We apply PreSTO to three MH-based samplers that target the same power distribution, and compare each sampler with its PreSTO variant:

- **PowerMH** (Karan & Du, 2026) constructs a continuation block by block. At each stage, it extends the current prefix using the proposal model and performs several MH updates targeting the power-sharpened distribution over the extended prefix. Each update selects a cut position uniformly at random, retains the preceding tokens, resamples the suffix from the proposal model, and accepts or rejects the candidate using the MH correction.
- **EntropyCut** (Zhou et al., 2026) modifies PowerMH by selecting cut positions according to increases in the base model's next-token entropy. These entropy jumps serve as a proxy for consequential reasoning decisions. The sampler regenerates the suffix from the selected position and applies the MH correction to preserve the power target. This baseline tests whether concentrating proposals at more informative cut positions improves sampling.
- **MultiTryMH** replaces each single-proposal PowerMH update with Multiple-Try Metropolis (MultiTryMH). We use a simplified version of Entropy-Guided Power Sampling (EGPS), following the "EGPS (MultiTryMH, with random pos)" variant in Figure 3 of Guo et al. (2026). Specifically, we omit entropy-guided cut-index sampling and sample the cut position uniformly at random.

  At the selected position, the sampler generates multiple candidate suffixes in parallel and assigns each an importance weight based on its MH ratio under the power-sharpened target. It then selects

a candidate with probability proportional to its weight and applies the MultiTryMH acceptance correction.
This baseline preserves the power target while considering multiple candidate suffixes within each MH transition.

D.2 Choice of Datasets

We evaluate the methods on seven datasets in three task categories, using the first 20 prompts of each dataset in its released order, without shuffling. Table 3 lists the datasets and their sizes.

- **Mathematical reasoning.** MATH500 (Lightman et al., 2024)[1] and AIME 24&25[2] assess problem solving in competitive mathematics. Our 20 AIME prompts are problems 1–15 of AIME 2024 I and problems 1–5 of AIME 2025 I.
- **Code generation.** HumanEval (Chen et al., 2021)[3], MBPP (Austin et al., 2021)[4], and LiveCodeBench (LCB V6) (Jain et al., 2025)[5] evaluate generated programs against test cases. A solution is counted as correct only if it passes all tests.
- **Scientific and general knowledge.** GPQA Diamond (Rein et al., 2024)[6] assesses advanced reasoning in physics, chemistry, and biology, and MMLU (Hendrycks et al., 2021a)[7] covers multiple-choice questions across 57 subjects. Because the test split is ordered by subject, our 20 MMLU prompts are the first 20 abstract algebra questions.

Table 3: Datasets and their sizes. We evaluate the first 20 prompts of each dataset in its released order (no shuffling), with a fixed response length of 1,024 tokens.

Category	Dataset	Instances
Mathematical reasoning	MATH500	500
	AIME 24&25	60
Code generation	HumanEval	164
	MBPP	257
	LCB V6	175
Scientific and general knowledge	GPQA Diamond	198
	MMLU	14,042

D.3 Choice of Base LLMs

We evaluate PreSTO on five base LLMs from the Qwen and Gemma families. Here, "base LLM" denotes the backbone p_0 used by PreSTO-PowerMH. We select these models based on recency and popularity: each is a recent open-weight release from a widely adopted model family. They also span two Qwen generations and parameter scales from 4B to 12B, allowing us to assess the method across a range of backbones.

- **Qwen3.** Qwen3-4B[8] and Qwen3-8B[9] provide two parameter scales from the previous Qwen generation.
- **Qwen3.5.** Qwen3.5-4B[10] and Qwen3.5-9B[11] provide two parameter scales within the same model generation.

[1] https://huggingface.co/datasets/HuggingFaceH4/MATH-500
[2] AIME24: https://huggingface.co/datasets/Maxwell-Jia/AIME_2024; AIME25: https://huggingface.co/datasets/MathArena/aime_2025
[3] https://huggingface.co/datasets/openai/openai_humaneval
[4] https://huggingface.co/datasets/google-research-datasets/mbpp
[5] https://huggingface.co/livecodebench
[6] https://huggingface.co/datasets/Idavidrein/gpqa
[7] https://huggingface.co/datasets/cais/mmlu
[8] https://huggingface.co/Qwen/Qwen3-4B
[9] https://huggingface.co/Qwen/Qwen3-8B
[10] https://huggingface.co/Qwen/Qwen3.5-4B
[11] https://huggingface.co/Qwen/Qwen3.5-9B

- **Gemma4-12B.** Gemma4-12B-it[12] extends the evaluation to a larger instruction-tuned backbone from Google. The EntropyCut comparisons use the base checkpoint, Gemma4-12B-base[13].

D.4 Evaluation Metrics

We evaluate PreSTO-PowerMH along three dimensions: execution efficiency, sample quality, and hardware-resource efficiency. Execution efficiency measures the cost of running the same MH sampler for a fixed number of transitions. Throughout, K denotes the number of realized MH transitions, counting both accepted and rejected proposals.

Execution Efficiency

- *MH transitions per proposal-model call.* For each proposal-model call, we record the number of MH transitions it realizes; higher values indicate greater chain progress per call. Trace figures show the mean and standard deviation, as specified in their captions. Throughout, $\pm$ values are standard deviations across the 20 prompts of a single seed.
- *Wall-clock time and speedup.* We record the wall-clock time per call and the cumulative model-call time needed to complete K realized MH transitions. The speedup is the time a sequential sampler (PowerMH or EntropyCut) needs to complete K transitions divided by the time its PreSTO variant needs for the same K, measured with the same timing boundary for both methods. Our acceleration measurements cover only model calls; end-to-end latency additionally includes subtree management, traversal, and other system overhead.

  For a paired run at fixed K and either method of the pair, the *number of calls* of the method is the number of timed model calls it issues to complete the K realized MH transitions, and the *time of the method per call* is the mean wall-clock duration of these calls within that run. Their product is the method's cumulative model-call time, so the speedup is

$$\text{Speedup} = \frac{\text{number of calls of the sampler} \times \text{time of the sampler per call}}{\text{number of calls of its PreSTO variant} \times \text{time of the variant per call}} \tag{23}$$

  Equivalently, the speedup is the reduction factor in the number of calls divided by the increase factor in the time per call. Equation 23 uses the complete call counts and durations of a single run; dividing separately averaged summaries, such as the per-panel means in our figures, need not recover this aggregate value. The speedups in Table 1 are descriptive ratios of reported mean times and carry no confidence intervals.

Sample Quality

- *Likelihood and confidence.* Prior work evaluates MCMC samplers for text by comparing their sampled distributions with reference distributions (Forristal et al., 2023; Du et al., 2024). For power sharpening, Karan & Du (2026) characterize sampled responses by their base-model likelihood and confidence, both normalized by sequence length; we adopt these two diagnostics.

  Let $\mathbf{x}_{1:n}^{(K)}$ denote the terminal state after exactly K realized MH transitions, where n is its length in tokens. Its length-normalized log-likelihood is

$$\text{log-likelihood}(\mathbf{x}^{(K)}) = \frac{1}{n} \log p_0(\mathbf{x}_{1:n}^{(K)} \mid \mathbf{x}_0), \tag{24}$$

  and its confidence is the average negative entropy of the base model's next-token distributions along the sequence:

$$\text{Confidence}(\mathbf{x}^{(K)}) = \frac{1}{n} \sum_{t=1}^{n} \sum_{u \in \mathcal{V}} p_0(u \mid \mathbf{x}_0, \mathbf{x}_{1:t-1}^{(K)}) \log p_0(u \mid \mathbf{x}_0, \mathbf{x}_{1:t-1}^{(K)}), \tag{25}$$

  where $\mathcal{V}$ is the vocabulary. Both statistics are evaluated under the same base model p_0, so they are comparable across samplers. A higher length-normalized log-likelihood indicates a higher average token log-probability under the base model, and a higher confidence indicates lower predictive uncertainty along the sequence.

  For each pairwise comparison, we match prompts and seeds, run both MH-based methods for the same K, and compute both statistics on their terminal states. Box plots summarize the resulting distributions across matched runs, and a paired TOST equivalence test assesses whether the two methods agree (Appendix E.3).

[12] https://huggingface.co/google/gemma-4-12B-it

[13] https://huggingface.co/google/gemma-4-12B

Hardware-Resource Efficiency

- *Peak KV-pool use.* Because vLLM preallocates the KV-cache block pool, reserved capacity does not reflect dynamic cache demand. We therefore report the peak fraction of occupied blocks in the pool; higher use indicates greater cache demand. Pool use is polled, so brief spikes may be missed. This metric captures dynamic KV-block demand only, so we also record peak total GPU memory (GiB) to compare overall memory use with sequential PowerMH.
- *Prefix-cache hit rate.* We measure prefix reuse as the cumulative token hit rate: the fraction of queried prefix tokens whose KV states are already cached. A higher hit rate indicates greater reuse of prefix computation.
- *Prefix-cache miss tokens per transition.* We divide the number of prefix-cache miss tokens by the number of realized MH transitions. This complements the hit rate by measuring the absolute uncached workload per transition.

D.5 Software and Hardware Configuration

Table 4 summarizes the experimental software and hardware environment. We implement PowerMH, EntropyCut, and their PreSTO variants on top of vLLM, using custom engine patches.

vLLM Engine patches. Every MH transition scores a proposed suffix under both the proposal LLM q_τ and the base LLM p_0. EntropyCut additionally needs the next-token entropy of p_0 at every position of the current sequence. Stock vLLM returns log-probabilities under only one distribution per request, taken either before or after logits processing depending on its configuration, so obtaining both $\log q_\tau$ and $\log p_0$ would require a second prefill pass over every proposal. We therefore patch the vLLM V1 engine to return these statistics from the same forward pass that generates each token. The patches subclass vLLM's own components and substitute them at runtime, so vLLM itself is not forked.

- *Sampler.* The patched sampler keeps a float32 copy of the logits before temperature scaling, penalties, or any other logits processor is applied. For each sampled token x_t, it returns three quantities. From the processed logits, it returns the proposal log-probability $\log q_\tau(x_t \mid \mathbf{x}_{0:t-1})$, as stock vLLM does. From the unprocessed copy, it returns the base-model log-probability $\log p_0(x_t \mid \mathbf{x}_{0:t-1})$ and the full-vocabulary entropy, in nats,

$$H_t = -\sum_{u \in \mathcal{V}} p_0(u \mid \mathbf{x}_{0:t-1}) \log p_0(u \mid \mathbf{x}_{0:t-1}), \tag{26}$$

  whose negated average over the sequence is the confidence in equation 25. For a configured set of temperatures, the sampler can also return the log-normalizer

$$\log Z_\tau(\mathbf{x}_{0:t-1}) = \log \sum_{u \in \mathcal{V}} p_0(u \mid \mathbf{x}_{0:t-1})^{1/\tau}. \tag{27}$$

  For any configured τ, the log-probability of x_t under the untruncated tempered proposal then follows without another engine request:

$$\log q_\tau(x_t \mid \mathbf{x}_{0:t-1}) = \frac{1}{\tau} \log p_0(x_t \mid \mathbf{x}_{0:t-1}) - \log Z_\tau(\mathbf{x}_{0:t-1}). \tag{28}$$

- *Output path.* A custom worker, GPU model runner, scheduler, engine-core output messages, and output processor carry these per-token values through vLLM's pipeline, including serialization between the engine-core and front-end processes. The values are attached to each request's completion output alongside the sampled tokens. The engine is configured to return processed log-probabilities, so the reported $\log q_\tau$ is that of the tempered proposal actually sampled from.

The patches leave scheduling, batching, KV-cache management, and attention kernels unchanged. PreSTO submits the proposals of a prefetched subtree as one batched generation call. The requests share their common prefixes through vLLM's automatic prefix caching, which is enabled for both PowerMH and PreSTO. An optional hook that evicts KV blocks held only by discarded proposals is disabled in all reported runs.

D.6 Hyperparameter Configuration

Table 5 lists the default power-sampling settings and the settings for our case studies and ablation studies. All runs generate a response of $n = 1024$ tokens as a single block, so the block size is

Table 4: Software and hardware configuration.

Software configuration

Software	Version / configuration
Python	3.13.0
PyTorch	2.13.0
vLLM	0.27.1 (V1 engine) with custom engine patches
Transformers	5.14.1

Hardware configuration

Hardware	Specification
GPU model	1× NVIDIA H100 96 GB (GH200 Grace Hopper Superchip)
CPU model	NVIDIA Grace CPU, 72 cores (3.456 GHz)
Host memory	480 GB LPDDR5X unified memory

$B = n = 1024$ tokens. Generation does not stop at the end-of-sequence token, so every state has exactly n tokens.

The proposal LLM q_τ samples from the base model at temperature τ. We use the power exponent $\alpha = 4.0$ with $\tau = 1/\alpha = 0.25$, following Karan & Du (2026).

Main experiments and case studies. Every reported run uses $K = 100$ realized MH transitions on the first 20 prompts of each dataset with seed 10086. The paired comparisons use the default BFS traversal with $N_{\text{pf}} = 20$, except $N_{\text{pf}} = 10$ for PreSTO-PowerMH on LCB V6, and EntropyCut uses an entropy threshold of 4.0 to select cut indices. Case Studies 1, 2, and 4 use Qwen3.5-9B on LCB V6; Case Study 3 uses Qwen3.5-4B on MATH500, AIME 24&25, GPQA, MMLU, and LCB V6. Case Study 2 sweeps $N_{\text{pf}} \in \{4, 8, 12, 16, 20\}$ for PreSTO-PowerMH while holding K fixed, so that each budget completes the same number of transitions; Case Study 4 uses $N_{\text{pf}} \in \{10, 20\}$ for PreSTO-PowerMH and $N_{\text{pf}} \in \{4, 8, 12, 20\}$ for PreSTO-EntropyCut.

Table 5: Default power-sampling settings (upper block) and the settings that the case studies change (lower block).

Parameter	Setting
Default power-sampling configuration	
Response length	$n = 1024$ tokens (fixed; no early stop at EOS)
Block size	$B = n = 1024$ tokens (one block)
Power exponent	$\alpha = 4.0$
Proposal LLM	q_τ = base model sampled at temperature τ
Proposal temperature	$\tau = 1/\alpha = 0.25$
Realized MH transitions	$K = 100$
Prompts per dataset	first 20
Seed	10086
Traversal rule	default BFS
Prefetch budget	$N_{\text{pf}} = 20$ ($N_{\text{pf}} = 10$ for PreSTO-PowerMH on LCB V6)
EntropyCut entropy threshold	4.0
Case study configuration	
Base model and dataset	p_0 = Qwen3.5-9B on LCB V6 (Case Studies 1, 2, 4) p_0 = Qwen3.5-4B on five datasets (Case Study 3)
Prefetch budget (Case Study 2)	$N_{\text{pf}} \in \{4, 8, 12, 16, 20\}$
Prefetch budget (Case Study 4)	$\{10, 20\}$ for PreSTO-PowerMH; $\{4, 8, 12, 20\}$ for PreSTO-EntropyCut

E Additional Experimental Results

This appendix reports the paired execution results behind Section 5 (Appendix E.1), the prefetch-budget traces (Appendix E.2), the log-likelihood and confidence tests (Appendix E.3), and the prediction of ϵ-certainty before generation (Appendix E.4). All pairs use $\alpha = 4$; we do not vary α, and preserving the MH kernel does not imply that prefetching efficiency is insensitive to it.

E.1 Paired execution results across base LLMs

The following tables and figures collect the available PowerMH and EntropyCut execution results with their PreSTO implementations, at the recorded traversal and prefetch-budget settings. The PowerMH comparisons use the instruction-tuned Gemma4-12B-it checkpoint on all seven datasets; the EntropyCut comparisons use the base checkpoint (Gemma4-12B-base), for which only MATH500 has complete runs of both samplers. The PowerMH comparisons use $N_{\mathrm{pf}} = 20$, except $N_{\mathrm{pf}} = 10$ on LCB V6, and the EntropyCut comparisons use $N_{\mathrm{pf}} = 20$ throughout. Every result runs to $K = 100$ realized transitions. Each three-panel trace figure reports cumulative model-call time, transitions per call, and seconds per call, with individual traces, means, and one-standard-deviation bands as indicated in its legend. Speedups divide the displayed mean cumulative times; the individual standard deviations do not supply confidence intervals for these ratios. Tables 6, 7, and 8 collect the PowerMH, EntropyCut, and MultiTryMH metrics, respectively, from the execution traces in the format $\mathrm{mean}_{\pm\mathrm{std}}$.

Figure 1 places all 92 settings by the two factors of the decomposition in equation 23, the reduction in the number of model calls and the increase in time per call, on log scales with the latter reversed; dashed diagonals are iso-speedup lines, and larger speedups lie toward the upper right. Case Study 1 in Section 5.2 discusses the ranges of both factors.

Default BFS traversal at a fixed prefetch budget. Figure 11 shows the per-call traces on MATH500 for all five base LLMs, with the default BFS traversal at $N_{\mathrm{pf}} = 20$. Traces for the other datasets are omitted; Table 6 reports their means and standard deviations, and Figure 1 places every setting. The Gemma4-12B-it runs contain 20 complete PowerMH and PreSTO-PowerMH traces per dataset.

EntropyCut comparisons. Figure 12 shows the EntropyCut and PreSTO-EntropyCut traces on MATH500 for all five base LLMs. Traces for the other datasets are omitted; Table 7 reports their means and standard deviations. The corresponding kernel-preservation check remains necessary: the PowerMH theorem alone does not verify the EntropyCut integration, and these execution measurements capture cost rather than correctness.

MultiTryMH baselines. Table 8 reports MultiTryMH and PreSTO-MultiTryMH in the format of Tables 6 and 7. PreSTO-MultiTryMH completes 1.75–2.11 MH transitions per model call and reduces the mean total time of sequential MultiTryMH by $1.8\times$–$2.8\times$ for the Qwen3 models and Gemma4-12B-it, and by $5.5\times$–$8.1\times$ for the Qwen3.5 models. Part of this reduction comes from the implementation rather than from prefetching: PreSTO-MultiTryMH reuses the sampled proposal log-probabilities, whereas sequential MultiTryMH rescores every candidate at all three temperatures. Where the prompt counts differ, the two runs also cover different prompts, so these speedups are descriptive. MultiTryMH draws four candidate suffixes per transition, each at a proposal temperature drawn uniformly from $\{0.25, 0.5, 1.0\}$, with uniform cuts, $\alpha = 4$, a single block of 1,024 tokens, and the same seed and first 20 prompts as the other samplers. For sequential MultiTryMH, 19 of the 30 available base LLM–dataset pairs have finished all 20 prompts, and MMLU has no runs; for PreSTO-MultiTryMH, 9 of 29 pairs have. At $K = 100$, MultiTryMH takes $1.6\times$–$6.7\times$ the cumulative time of PowerMH in Table 6: $1.6\times$–$2.6\times$ for the Qwen3 models and Gemma4-12B-it, and $5.8\times$–$6.7\times$ for the Qwen3.5 models, excluding LCB V6 with Qwen3.5-9B, whose PowerMH baseline was run on LCB V6. The samplers ran on different dates and machines, so these ratios are descriptive.

E.2 Prefetch-budget traces

Prefetch budget under PowerMH and EntropyCut. On LCB V6 with Qwen3.5-9B under the default BFS traversal, we sweep $N_{\mathrm{pf}} \in \{4, 6, \ldots, 20\}$ for PreSTO-PowerMH and $N_{\mathrm{pf}} \in \{4, 8, 12, 20\}$ for PreSTO-EntropyCut; Table 2 reports the PreSTO-PowerMH budgets $\{4, 8, 12, 16, 20\}$. Both samplers respond to the budget in almost the same way. Realized transitions per call grow from 2.45 (PreSTO-PowerMH) and 2.38 (PreSTO-EntropyCut) at $N_{\mathrm{pf}} = 4$ to 4.12 for both at $N_{\mathrm{pf}} = 20$, while each call slows from 8.3 to 10.8 seconds (PreSTO-PowerMH) and from 7.8 to 11.0 seconds

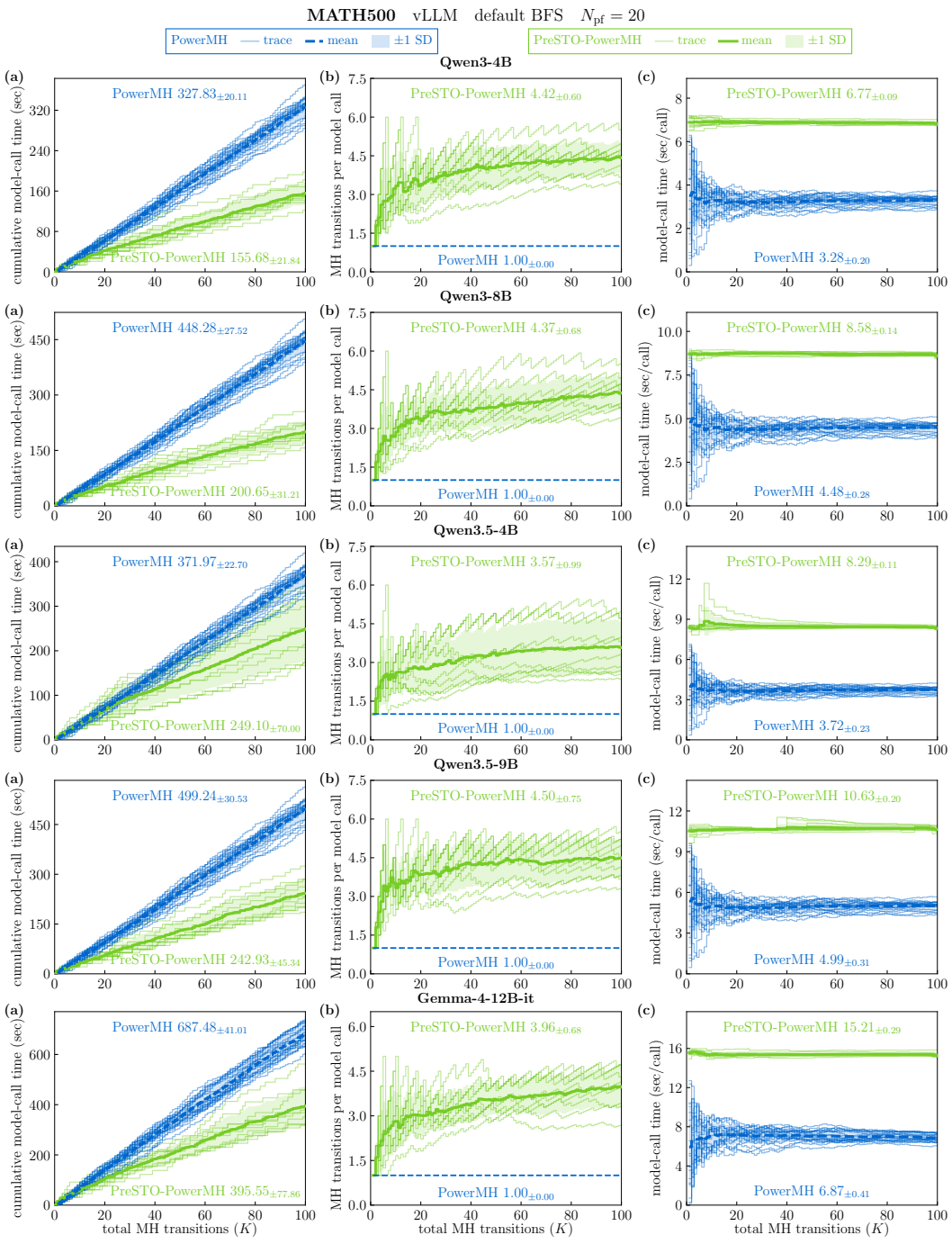


Figure 11: **PowerMH and PreSTO-PowerMH on MATH500.** Rows show the five labeled base LLMs at $N_{\text{pf}} = 20$ and $K = 100$.

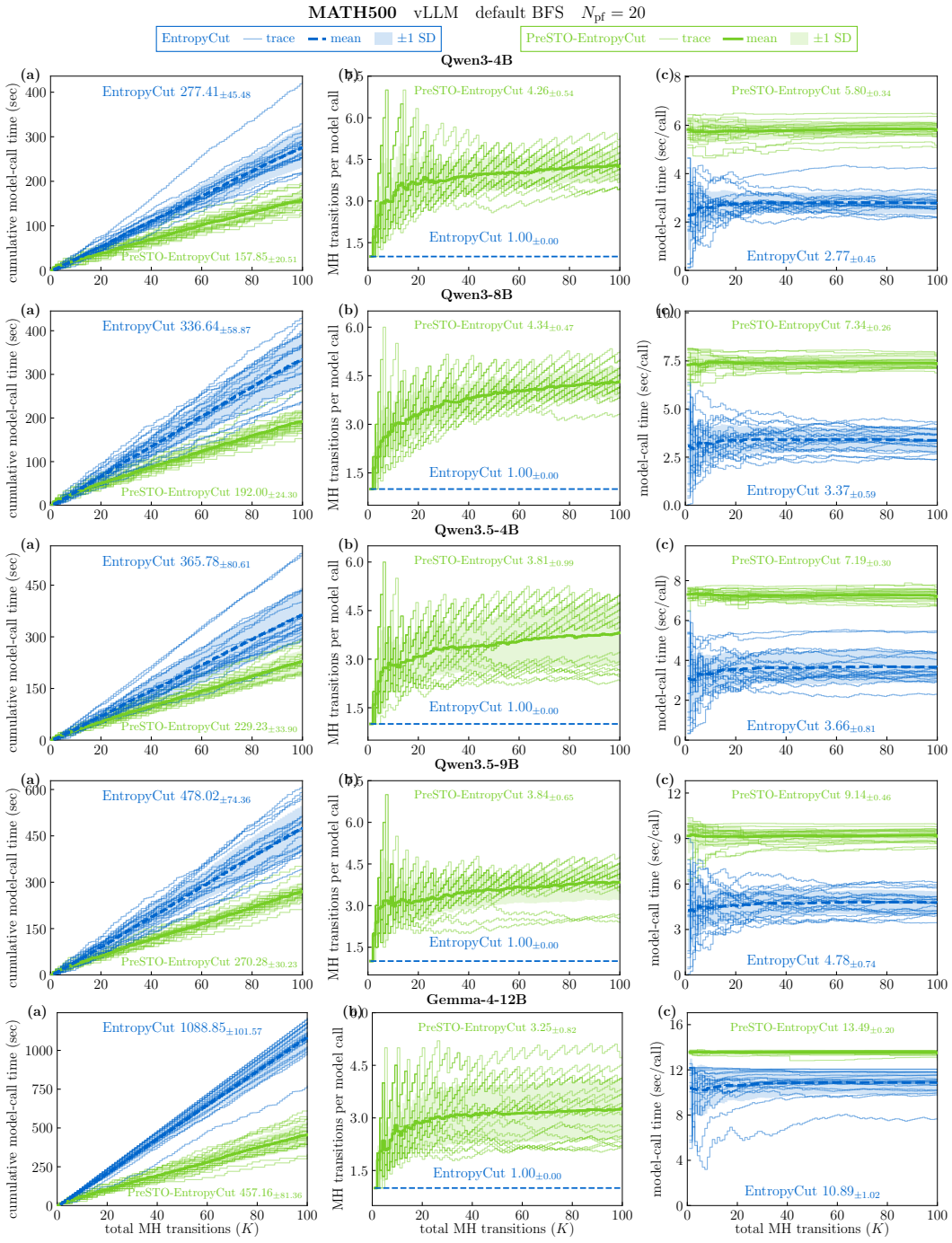


Figure 12: **EntropyCut and PreSTO-EntropyCut on MATH500.** Rows show Qwen3-4B, Qwen3-8B, Qwen3.5-4B, Qwen3.5-9B, and Gemma4-12B-base with the default BFS traversal at $N_{\text{pf}} = 20$ and $K = 100$. Each row reports cumulative model-call time, transitions per call, and seconds per call.

Table 6: PowerMH execution metrics across base LLMs. Cells show PowerMH / PreSTO-PowerMH as $\mathrm{mean}_{\pm\mathrm{std}}$ at $K = 100$, with the default BFS traversal at $N_{\mathrm{pf}} = 20$ ($N_{\mathrm{pf}} = 10$ on LCB V6); † marks a run on LCB V6.

Dataset Base LLM	Total Time (s) ↓	Transitions per call ↑	Time/call (s) ↓	Speedup
MATH500 (Fig. 11)				
Qwen3-4B	$327.83_{\pm 20.11}/155.68_{\pm 21.84}$	$1/4.42_{\pm 0.60}$	$3.28_{\pm 0.20}/6.77_{\pm 0.09}$	2.11×
Qwen3-8B	$448.28_{\pm 27.52}/200.65_{\pm 31.21}$	$1/4.37_{\pm 0.68}$	$4.48_{\pm 0.28}/8.58_{\pm 0.14}$	2.23×
Qwen3.5-4B	$371.97_{\pm 22.70}/249.10_{\pm 70.00}$	$1/3.57_{\pm 0.99}$	$3.72_{\pm 0.23}/8.29_{\pm 0.11}$	1.49×
Qwen3.5-9B	$499.24_{\pm 30.53}/242.93_{\pm 45.34}$	$1/4.50_{\pm 0.75}$	$4.99_{\pm 0.31}/10.63_{\pm 0.20}$	2.06×
Gemma4-12B-it	$687.48_{\pm 41.01}/395.55_{\pm 77.86}$	$1/3.96_{\pm 0.68}$	$6.87_{\pm 0.41}/15.21_{\pm 0.29}$	1.74×
AIME 24&25				
Qwen3-4B	$318.76_{\pm 19.59}/150.66_{\pm 32.25}$	$1/4.66_{\pm 0.86}$	$3.19_{\pm 0.20}/6.78_{\pm 0.11}$	2.12×
Qwen3-8B	$447.87_{\pm 27.50}/195.48_{\pm 34.63}$	$1/4.48_{\pm 0.71}$	$4.48_{\pm 0.28}/8.53_{\pm 0.15}$	2.29×
Qwen3.5-4B	$360.89_{\pm 22.43}/217.57_{\pm 37.98}$	$1/3.99_{\pm 0.62}$	$3.61_{\pm 0.22}/8.47_{\pm 0.16}$	1.66×
Qwen3.5-9B	$498.37_{\pm 30.48}/239.31_{\pm 44.47}$	$1/4.56_{\pm 0.72}$	$4.98_{\pm 0.30}/10.63_{\pm 0.31}$	2.08×
Gemma4-12B-it	$686.87_{\pm 38.96}/389.43_{\pm 57.27}$	$1/3.93_{\pm 0.53}$	$6.87_{\pm 0.39}/15.04_{\pm 0.23}$	1.76×
MBPP				
Qwen3-4B	$327.45_{\pm 20.07}/296.53_{\pm 65.99}$	$1/2.42_{\pm 0.65}$	$3.27_{\pm 0.20}/6.81_{\pm 0.08}$	1.10×
Qwen3-8B	$447.85_{\pm 27.46}/446.25_{\pm 41.30}$	$1/1.96_{\pm 0.19}$	$4.48_{\pm 0.27}/8.68_{\pm 0.07}$	1.00×
Qwen3.5-4B	$370.03_{\pm 22.56}/221.28_{\pm 69.63}$	$1/4.13_{\pm 0.94}$	$3.70_{\pm 0.23}/8.56_{\pm 0.30}$	1.67×
Qwen3.5-9B	$497.47_{\pm 30.41}/309.52_{\pm 88.73}$	$1/3.74_{\pm 0.84}$	$4.97_{\pm 0.30}/10.92_{\pm 0.32}$	1.61×
Gemma4-12B-it	$684.80_{\pm 38.84}/475.98_{\pm 177.87}$	$1/3.64_{\pm 1.29}$	$6.85_{\pm 0.39}/15.29_{\pm 0.15}$	1.44×
GPQA				
Qwen3-4B	$323.68_{\pm 21.46}/198.61_{\pm 42.79}$	$1/3.55_{\pm 0.59}$	$3.24_{\pm 0.21}/6.83_{\pm 0.08}$	1.63×
Qwen3-8B	$442.49_{\pm 29.28}/208.59_{\pm 77.96}$	$1/4.40_{\pm 0.91}$	$4.42_{\pm 0.29}/8.55_{\pm 0.12}$	2.12×
Qwen3.5-4B	$365.82_{\pm 24.10}/217.07_{\pm 79.59}$	$1/4.16_{\pm 0.92}$	$3.66_{\pm 0.24}/8.38_{\pm 0.20}$	1.69×
Qwen3.5-9B	$491.59_{\pm 32.48}/239.92_{\pm 79.83}$	$1/4.73_{\pm 1.12}$	$4.92_{\pm 0.32}/10.57_{\pm 0.25}$	2.05×
Gemma4-12B-it	$687.19_{\pm 39.51}/411.00_{\pm 127.19}$	$1/3.90_{\pm 0.81}$	$6.87_{\pm 0.40}/15.12_{\pm 0.21}$	1.67×
HumanEval				
Qwen3-4B	$327.25_{\pm 20.07}/184.93_{\pm 54.81}$	$1/3.89_{\pm 1.04}$	$3.27_{\pm 0.20}/6.70_{\pm 0.11}$	1.77×
Qwen3-8B	$447.42_{\pm 27.44}/247.81_{\pm 73.78}$	$1/3.72_{\pm 0.95}$	$4.47_{\pm 0.27}/8.59_{\pm 0.09}$	1.81×
Qwen3.5-4B	$369.89_{\pm 22.61}/231.97_{\pm 54.39}$	$1/3.79_{\pm 0.92}$	$3.70_{\pm 0.23}/8.36_{\pm 0.24}$	1.59×
Qwen3.5-9B	$498.37_{\pm 30.48}/412.74_{\pm 78.17}$	$1/2.67_{\pm 0.46}$	$4.98_{\pm 0.30}/10.72_{\pm 0.21}$	1.21×
Gemma4-12B-it	$687.16_{\pm 39.54}/400.87_{\pm 82.35}$	$1/3.93_{\pm 0.74}$	$6.87_{\pm 0.40}/15.19_{\pm 0.17}$	1.71×
MMLU				
Qwen3-4B	$327.51_{\pm 20.06}/205.84_{\pm 62.37}$	$1/3.59_{\pm 1.06}$	$3.28_{\pm 0.20}/6.82_{\pm 0.11}$	1.59×
Qwen3-8B	$447.91_{\pm 27.44}/298.07_{\pm 123.34}$	$1/3.29_{\pm 1.17}$	$4.48_{\pm 0.27}/8.57_{\pm 0.13}$	1.50×
Qwen3.5-4B	$370.51_{\pm 22.60}/210.37_{\pm 47.85}$	$1/4.12_{\pm 0.81}$	$3.71_{\pm 0.23}/8.32_{\pm 0.17}$	1.76×
Qwen3.5-9B	$493.44_{\pm 31.14}/418.01_{\pm 122.83}$	$1/2.72_{\pm 0.78}$	$4.93_{\pm 0.31}/10.52_{\pm 0.15}$	1.18×
Gemma4-12B-it	$686.89_{\pm 39.47}/452.21_{\pm 61.30}$	$1/3.40_{\pm 0.61}$	$6.87_{\pm 0.39}/15.03_{\pm 0.21}$	1.52×
LCB V6				
Qwen3-4B	$318.79_{\pm 19.56}/182.05_{\pm 32.65}$	$1/3.68_{\pm 0.50}$	$3.19_{\pm 0.20}/6.55_{\pm 0.13}$	1.75×
Qwen3-8B	$432.26_{\pm 25.14}/260.60_{\pm 39.73}$	$1/3.40_{\pm 0.50}$	$4.32_{\pm 0.25}/8.68_{\pm 0.11}$	1.66×
Qwen3.5-4B	$368.07_{\pm 20.61}/210.89_{\pm 33.22}$	$1/3.58_{\pm 0.44}$	$3.68_{\pm 0.21}/7.43_{\pm 0.10}$	1.75×
Qwen3.5-9B	$481.02_{\pm 27.14}/253.29_{\pm 23.12}$	$1/3.86_{\pm 0.33}$	$4.81_{\pm 0.27}/9.71_{\pm 0.24}$	1.90×†
Gemma4-12B-it	$685.34_{\pm 39.07}/378.69_{\pm 37.47}$	$1/3.65_{\pm 0.35}$	$6.85_{\pm 0.39}/13.72_{\pm 0.18}$	1.81×

(PreSTO-EntropyCut). The resulting speedups appear in Table 2 for PreSTO-PowerMH and in the BFS bars of Figure 7 for PreSTO-EntropyCut.

Traces. Figure 13 shows the traces of every swept budget over the $K = 100$ realized MH transitions, with one curve per prefetch budget. Each PreSTO run is paired with the sequential run of identical settings, and a budget recorded more than once uses its latest run. All PowerMH budgets share one sequential baseline of 20 traces. The EntropyCut budgets were recorded on two dates, so the EntropyCut curve pools the 40 sequential traces from those dates.

Table 7: **EntropyCut execution metrics across base LLMs.** Cells show EntropyCut / PreSTO-EntropyCut as $\mathrm{mean}_{\pm\mathrm{std}}$ at $K = 100$. All runs use the default BFS traversal at $N_{\mathrm{pf}} = 20$.

Dataset Base LLM	Total Time (s) ↓	Transitions per call ↑	Time/call (s) ↓	Speedup
MATH500 (Fig. 12)				
Qwen3-4B	$277.41_{\pm 45.48}/157.85_{\pm 20.51}$	$1/4.26_{\pm 0.54}$	$2.77_{\pm 0.45}/5.80_{\pm 0.34}$	1.76×
Qwen3-8B	$336.64_{\pm 58.87}/192.00_{\pm 24.30}$	$1/4.34_{\pm 0.47}$	$3.37_{\pm 0.59}/7.34_{\pm 0.26}$	1.75×
Qwen3.5-4B	$365.78_{\pm 80.61}/229.23_{\pm 33.90}$	$1/3.81_{\pm 0.99}$	$3.66_{\pm 0.81}/7.19_{\pm 0.30}$	1.60×
Qwen3.5-9B	$478.02_{\pm 74.36}/270.28_{\pm 30.23}$	$1/3.84_{\pm 0.65}$	$4.78_{\pm 0.74}/9.14_{\pm 0.46}$	1.77×
Gemma4-12B-base	$1088.85_{\pm 101.57}/457.16_{\pm 81.36}$	$1/3.25_{\pm 0.82}$	$10.89_{\pm 1.02}/13.49_{\pm 0.20}$	2.38×
AIME 24&25				
Qwen3-4B	$255.04_{\pm 37.63}/150.54_{\pm 16.85}$	$1/4.57_{\pm 0.49}$	$2.55_{\pm 0.38}/6.08_{\pm 0.19}$	1.69×
Qwen3-8B	$345.86_{\pm 62.77}/192.57_{\pm 26.84}$	$1/4.37_{\pm 0.56}$	$3.46_{\pm 0.63}/7.43_{\pm 0.29}$	1.80×
Qwen3.5-4B	$405.21_{\pm 104.47}/196.78_{\pm 41.26}$	$1/4.29_{\pm 0.64}$	$4.05_{\pm 1.04}/7.03_{\pm 0.54}$	2.06×
Qwen3.5-9B	$490.26_{\pm 82.75}/277.80_{\pm 65.23}$	$1/3.83_{\pm 0.75}$	$4.90_{\pm 0.83}/9.06_{\pm 0.94}$	1.76×
MBPP				
Qwen3-4B	$260.73_{\pm 60.29}/197.46_{\pm 57.84}$	$1/3.51_{\pm 1.25}$	$2.61_{\pm 0.60}/6.26_{\pm 0.53}$	1.32×
Qwen3-8B	$367.12_{\pm 82.44}/249.51_{\pm 86.94}$	$1/4.00_{\pm 1.42}$	$3.67_{\pm 0.82}/8.51_{\pm 0.15}$	1.47×
Qwen3.5-4B	$509.39_{\pm 79.35}/207.31_{\pm 22.63}$	$1/4.13_{\pm 0.37}$	$5.09_{\pm 0.79}/7.62_{\pm 0.19}$	2.46×
Qwen3.5-9B	$762.07_{\pm 70.82}/267.03_{\pm 54.05}$	$1/4.30_{\pm 0.92}$	$7.62_{\pm 0.71}/9.90_{\pm 0.17}$	2.85×
GPQA				
Qwen3-4B	$372.97_{\pm 69.04}/178.64_{\pm 32.15}$	$1/4.30_{\pm 0.67}$	$3.73_{\pm 0.69}/6.64_{\pm 0.25}$	2.09×
Qwen3-8B	$377.49_{\pm 64.81}/210.70_{\pm 24.84}$	$1/4.39_{\pm 0.50}$	$3.77_{\pm 0.65}/8.21_{\pm 0.41}$	1.79×
Qwen3.5-4B	$397.44_{\pm 97.55}/203.25_{\pm 52.71}$	$1/4.36_{\pm 0.77}$	$3.97_{\pm 0.98}/7.38_{\pm 0.52}$	1.96×
Qwen3.5-9B	$529.97_{\pm 101.82}/269.92_{\pm 75.09}$	$1/4.21_{\pm 0.75}$	$5.30_{\pm 1.02}/9.58_{\pm 0.81}$	1.96×
HumanEval				
Qwen3-4B	$380.83_{\pm 108.02}/203.08_{\pm 60.14}$	$1/3.68_{\pm 1.37}$	$3.81_{\pm 1.08}/5.89_{\pm 0.88}$	1.88×
Qwen3-8B	$441.98_{\pm 154.66}/215.14_{\pm 88.83}$	$1/4.44_{\pm 1.79}$	$4.42_{\pm 1.55}/7.30_{\pm 1.52}$	2.05×
Qwen3.5-4B	$435.56_{\pm 105.27}/211.17_{\pm 46.11}$	$1/3.71_{\pm 0.76}$	$4.36_{\pm 1.05}/6.49_{\pm 1.17}$	2.06×
Qwen3.5-9B	$767.76_{\pm 90.41}/268.95_{\pm 108.36}$	$1/4.78_{\pm 2.47}$	$7.68_{\pm 0.90}/9.58_{\pm 0.35}$	2.85×
MMLU				
Qwen3-4B	$390.39_{\pm 105.60}/192.26_{\pm 42.67}$	$1/4.01_{\pm 0.73}$	$3.90_{\pm 1.06}/6.60_{\pm 0.34}$	2.03×
Qwen3-8B	$612.03_{\pm 184.09}/278.39_{\pm 93.41}$	$1/3.90_{\pm 1.15}$	$6.12_{\pm 1.84}/8.87_{\pm 0.69}$	2.20×
Qwen3.5-4B	$395.79_{\pm 86.46}/228.59_{\pm 49.51}$	$1/3.75_{\pm 0.83}$	$3.96_{\pm 0.86}/7.31_{\pm 0.59}$	1.73×
Qwen3.5-9B	$841.38_{\pm 140.68}/352.80_{\pm 95.07}$	$1/3.49_{\pm 0.87}$	$8.41_{\pm 1.41}/10.56_{\pm 0.44}$	2.38×
LCB V6				
Qwen3-4B	$250.62_{\pm 56.09}/151.82_{\pm 26.30}$	$1/4.19_{\pm 0.62}$	$2.51_{\pm 0.56}/5.38_{\pm 0.30}$	1.65×
Qwen3-8B	$340.08_{\pm 82.22}/220.38_{\pm 65.17}$	$1/3.92_{\pm 0.90}$	$3.40_{\pm 0.82}/7.31_{\pm 0.81}$	1.54×
Qwen3.5-4B	$337.14_{\pm 37.91}/198.35_{\pm 48.78}$	$1/4.31_{\pm 0.59}$	$3.37_{\pm 0.38}/7.13_{\pm 0.96}$	1.70×
Qwen3.5-9B	$463.80_{\pm 52.32}/242.89_{\pm 35.93}$	$1/4.22_{\pm 0.58}$	$4.64_{\pm 0.52}/9.00_{\pm 0.49}$	1.91×

For every budget, the cumulative model-call time (panel a) separates from the sequential baseline within the first few transitions, and the gap widens steadily. Transitions per call (panel b) settle after about 20–40 transitions, and the settled level generally rises with N_{pf}. Time per call (panel c) stays roughly constant along the chain for each budget, so the budget, not the position in the chain, sets the per-call latency.

E.3 Log-likelihood and confidence

Following Karan & Du (2026), we describe terminal traces by their token-normalized log-likelihood equation 24 and confidence equation 25 under p_0.

Settings. We compare each sampler with its PreSTO variant on matched dataset–base-LLM pairs: PowerMH against PreSTO-PowerMH with uniform cuts ($N_{\mathrm{pf}} = 20$; $N_{\mathrm{pf}} = 10$ on LCB V6), and EntropyCut against PreSTO-EntropyCut ($N_{\mathrm{pf}} = 20$). Within a pair, all runs share the prompts, prompt format, $\alpha = 4$, $K = 100$, seed, and a single 1,024-token block, on the first 20 prompts of each benchmark. The samplers store decoded responses, so we rescore every terminal trace under p_0 with one teacher-forced forward pass (bfloat16 weights, fp32 log-softmax over the full vocabulary). Decoding drops special tokens, and PowerMH and EntropyCut continue past the end-of-sequence token within their fixed horizon, so a re-encoded response can differ from the sampled

Table 8: **MultiTryMH execution metrics across base LLMs.** Cells show MultiTryMH / PreSTO-MultiTryMH as $\mathrm{mean}_{\pm\mathrm{std}}$ at $K = 100$; PreSTO-MultiTryMH uses the default BFS traversal with $N_{\mathrm{pf}} = 20$ suffix requests per call. MultiTryMH times are cumulative MH-step time, covering candidate generation and rescoring, and each MH step counts as one call; PreSTO-MultiTryMH times are summed model-call time, and its implementation reuses the sampled proposal log-probabilities instead of rescoring candidates. Speedup is the ratio of mean total times. Rows where either run has fewer than 20 finished prompts list the counts n as MultiTryMH/PreSTO-MultiTryMH; the two runs then cover different prompts. Dashes mark unavailable measurements or speedups.

Dataset Base LLM	Total Time (s) ↓	Transitions per call ↑	Time/call (s) ↓	Speedup
MATH500				
Qwen3-4B (n=20/0)	$633.12_{\pm 30.06}$/–	1/–	$6.33_{\pm 0.30}$/–	–
Qwen3-8B	$724.48_{\pm 34.92}$/$381.42_{\pm 19.47}$	$1/2.04_{\pm 0.10}$	$7.24_{\pm 0.35}$/$7.75_{\pm 0.17}$	1.90×
Qwen3.5-4B (n=19/20)	$2157.13_{\pm 152.58}$/$376.85_{\pm 14.87}$	$1/2.05_{\pm 0.08}$	$21.57_{\pm 1.53}$/$7.72_{\pm 0.16}$	5.72×
Qwen3.5-9B (n=12/20)	$3346.52_{\pm 177.47}$/$468.25_{\pm 25.00}$	$1/2.05_{\pm 0.11}$	$33.47_{\pm 1.77}$/$9.58_{\pm 0.17}$	7.15×
Gemma4-12B-it (n=20/15)	$1488.39_{\pm 77.80}$/$685.54_{\pm 33.90}$	$1/1.85_{\pm 0.09}$	$14.88_{\pm 0.78}$/$12.66_{\pm 0.20}$	2.17×
AIME 24&25				
Qwen3-4B	$633.32_{\pm 31.36}$/$304.52_{\pm 17.52}$	$1/2.03_{\pm 0.11}$	$6.33_{\pm 0.31}$/$6.16_{\pm 0.18}$	2.08×
Qwen3-8B (n=20/19)	$867.08_{\pm 41.03}$/$390.46_{\pm 15.86}$	$1/1.99_{\pm 0.06}$	$8.67_{\pm 0.41}$/$7.74_{\pm 0.16}$	2.22×
Qwen3.5-4B (n=20/0)	$2337.06_{\pm 148.90}$/–	1/–	$23.37_{\pm 1.49}$/–	–
Qwen3.5-9B (n=13/15)	$3238.57_{\pm 227.85}$/$474.88_{\pm 23.85}$	$1/2.05_{\pm 0.10}$	$32.39_{\pm 2.28}$/$9.73_{\pm 0.23}$	6.82×
Gemma4-12B-it (n=20/17)	$1705.50_{\pm 87.72}$/$605.42_{\pm 23.91}$	$1/2.07_{\pm 0.08}$	$17.06_{\pm 0.88}$/$12.52_{\pm 0.30}$	2.82×
MBPP				
Qwen3-4B	$747.32_{\pm 38.12}$/$304.20_{\pm 14.70}$	$1/2.00_{\pm 0.10}$	$7.47_{\pm 0.38}$/$6.08_{\pm 0.13}$	2.46×
Qwen3-8B	$857.29_{\pm 42.09}$/$395.39_{\pm 26.91}$	$1/1.96_{\pm 0.12}$	$8.57_{\pm 0.42}$/$7.73_{\pm 0.18}$	2.17×
Qwen3.5-4B (n=19/20)	$2287.97_{\pm 127.21}$/$391.29_{\pm 22.63}$	$1/1.97_{\pm 0.13}$	$22.88_{\pm 1.27}$/$7.69_{\pm 0.15}$	5.85×
Qwen3.5-9B (n=14/13)	$3434.32_{\pm 219.55}$/$512.45_{\pm 25.92}$	$1/1.87_{\pm 0.09}$	$34.34_{\pm 2.20}$/$9.56_{\pm 0.11}$	6.70×
Gemma4-12B-it (n=20/10)	$1695.70_{\pm 84.39}$/$695.54_{\pm 32.35}$	$1/1.81_{\pm 0.09}$	$16.96_{\pm 0.84}$/$12.58_{\pm 0.26}$	2.44×
GPQA				
Qwen3-4B	$643.32_{\pm 30.05}$/$307.49_{\pm 18.08}$	$1/1.97_{\pm 0.13}$	$6.43_{\pm 0.30}$/$6.05_{\pm 0.15}$	2.09×
Qwen3-8B (n=20/17)	$873.76_{\pm 43.90}$/$401.99_{\pm 25.59}$	$1/2.00_{\pm 0.14}$	$8.74_{\pm 0.44}$/$8.00_{\pm 0.19}$	2.17×
Qwen3.5-4B (n=20/18)	$2119.49_{\pm 128.37}$/$383.31_{\pm 15.26}$	$1/2.04_{\pm 0.10}$	$21.19_{\pm 1.28}$/$7.82_{\pm 0.15}$	5.53×
Qwen3.5-9B (n=13/15)	$3168.57_{\pm 188.02}$/$467.79_{\pm 21.67}$	$1/2.04_{\pm 0.10}$	$31.69_{\pm 1.88}$/$9.51_{\pm 0.18}$	6.77×
Gemma4-12B-it (n=20/10)	$1457.96_{\pm 70.28}$/$680.50_{\pm 28.42}$	$1/1.84_{\pm 0.08}$	$14.58_{\pm 0.70}$/$12.49_{\pm 0.23}$	2.14×
HumanEval				
Qwen3-4B	$612.09_{\pm 29.96}$/$316.31_{\pm 21.62}$	$1/1.94_{\pm 0.14}$	$6.12_{\pm 0.30}$/$6.12_{\pm 0.18}$	1.94×
Qwen3-8B (n=20/11)	$728.37_{\pm 35.45}$/$403.92_{\pm 33.98}$	$1/1.92_{\pm 0.17}$	$7.28_{\pm 0.35}$/$7.71_{\pm 0.16}$	1.80×
Qwen3.5-4B (n=19/17)	$2154.79_{\pm 108.22}$/$394.91_{\pm 23.18}$	$1/1.96_{\pm 0.13}$	$21.55_{\pm 1.08}$/$7.71_{\pm 0.18}$	5.46×
Qwen3.5-9B (n=11/8)	$3323.84_{\pm 185.75}$/$530.38_{\pm 33.03}$	$1/1.80_{\pm 0.12}$	$33.24_{\pm 1.86}$/$9.52_{\pm 0.18}$	6.27×
Gemma4-12B-it (n=20/4)	$1459.11_{\pm 71.13}$/$671.32_{\pm 8.55}$	$1/1.88_{\pm 0.04}$	$14.59_{\pm 0.71}$/$12.61_{\pm 0.33}$	2.17×
LCB V6				
Qwen3-4B (n=20/9)	$661.84_{\pm 43.62}$/$320.16_{\pm 7.51}$	$1/1.95_{\pm 0.08}$	$6.62_{\pm 0.44}$/$6.24_{\pm 0.15}$	2.07×
Qwen3-8B (n=20/6)	$908.99_{\pm 62.75}$/$400.61_{\pm 21.75}$	$1/1.97_{\pm 0.13}$	$9.09_{\pm 0.63}$/$7.89_{\pm 0.17}$	2.27×
Qwen3.5-4B (n=17/6)	$2177.28_{\pm 194.11}$/$374.97_{\pm 14.71}$	$1/2.05_{\pm 0.12}$	$21.77_{\pm 1.94}$/$7.66_{\pm 0.16}$	5.81×
Qwen3.5-9B (n=10/2)	$3701.78_{\pm 331.75}$/$455.59_{\pm 3.08}$	$1/2.11_{\pm 0.03}$	$37.02_{\pm 3.32}$/$9.59_{\pm 0.08}$	8.13×
Gemma4-12B-it (n=17/1)	$1775.58_{\pm 107.30}$/$719.49_{\pm 0.00}$	$1/1.75_{\pm 0.00}$	$17.76_{\pm 1.07}$/$12.62_{\pm 0.00}$	2.47×

token sequence; we keep only the prompts whose re-encoded response has the sampled length in every run of the pair. Where PreSTO-EntropyCut also recorded both statistics during sampling, they check the rescoring: all 350 length-matched traces on MATH500, AIME 24&25, GPQA, and LCB V6 with Qwen3.5-4B and Qwen3.5-9B, and on MMLU with Qwen3.5-4B, agree within 0.01 (median absolute difference ≈ 0.001 in log-likelihood). For HumanEval with Qwen3 and Gemma4-12B-base, MBPP with Qwen3.5, and GPQA with Gemma4-12B-base, 46–81% of the traces disagree by more than 0.01 even at matching length, because re-encoding decoded text, especially code, can split whitespace into different tokens. We keep only the dataset–base-LLM combinations that pass this check and retain at least 10 prompts, which also excludes HumanEval with Qwen3.5. This leaves 9 EntropyCut pairs and 8 PowerMH pairs, the same combinations except LCB V6 with Qwen3.5-9B (Figure 14).

Statistical tests. For each pair and statistic we test the per-prompt paired difference, PreSTO minus baseline, for equivalence. Because failing to reject a difference does not show that two samplers agree, we use a paired TOST equivalence test (two one-sided t-tests): $p_{\mathrm{eq}} < 0.05$ shows that the

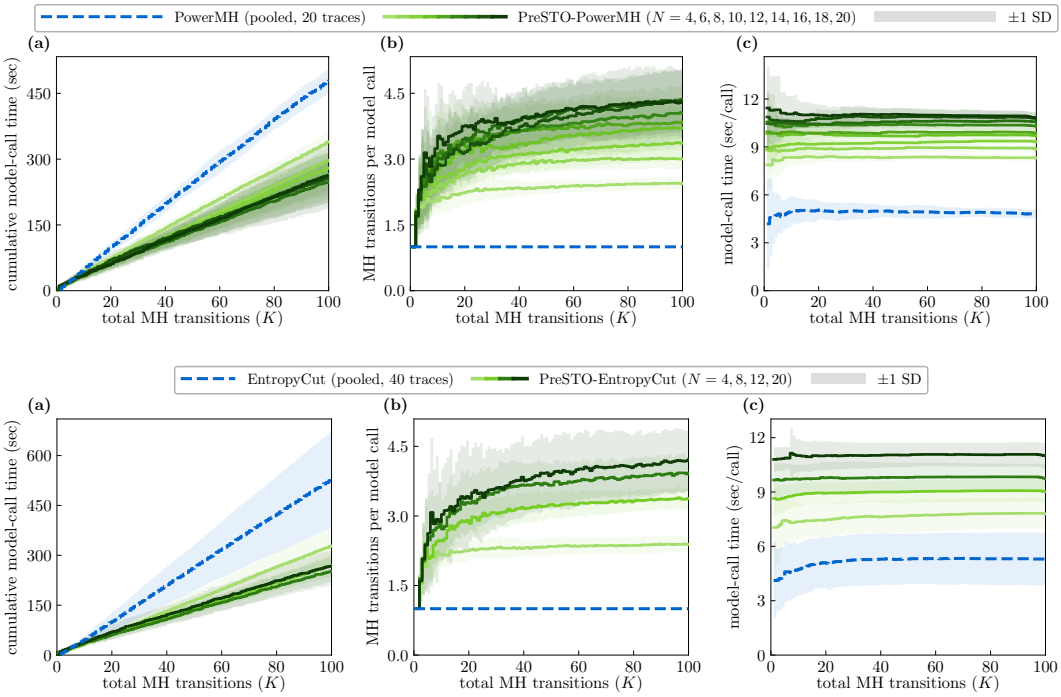


Figure 13: **Per-transition traces across prefetch budgets on LCB V6 with Qwen3.5-9B.** The runs use the default BFS traversal. Top: PowerMH against PreSTO-PowerMH; bottom: EntropyCut against PreSTO-EntropyCut. (a) Cumulative model-call time, (b) running MH transitions per model call, and (c) time per model call, against the number of realized MH transitions K. Solid curves show the PreSTO mean for each budget, from light (small N_{pf}) to dark (large N_{pf}). Dashed curves show the pooled sequential sampler, and shaded bands show ± 1 SD over traces.

mean difference lies within $\pm\delta$, with the margin fixed at $\delta = 0.2$ standard deviations of the baseline across prompts, a conventional small effect. For each sampler we also pool the standardized differences (difference divided by that standard deviation) over its pairs with a fixed-effect average and test the pooled difference against the same margin.

Results. For most pairs and when pooled, the differences between each sampler and its PreSTO variant lie within the equivalence margin; the pooled PowerMH differences are small, but their 90% intervals exclude zero.

- *Means nearly coincide.* The median gap between the PreSTO and baseline means is 0.002 in log-likelihood and 0.004–0.007 in confidence, against across-prompt standard deviations of about 0.05 and 0.10 (Figures 6 and 14).
- *Equivalence holds when pooled.* For both samplers and both statistics, the pooled difference lies within the ± 0.2 SD margin ($p_{\mathrm{eq}} < 0.001$ each). The pooled Δ/SD with 90% interval is $+0.038$ $[+0.010, +0.066]$ in log-likelihood and $+0.061$ $[+0.033, +0.088]$ in confidence for PowerMH, and $+0.025$ $[-0.002, +0.053]$ and $+0.017$ $[-0.009, +0.042]$ for EntropyCut.
- *Equivalence holds for most pairs.* It holds in 17 of 18 EntropyCut tests and 12 of 16 PowerMH tests (shaded rows in Figure 14). The exceptions are AIME with Qwen3.5-4B (both statistics, PreSTO slightly higher) and three near misses ($p_{\mathrm{eq}} \leq 0.104$).

Both statistics are compared within each base LLM, because changing p_0 changes the scoring distribution itself.

E.4 Predicting ϵ-certainty before generation

The predictive traversal rule (Appendix C.3) needs the class probabilities $\widehat{P}(A \leq \epsilon \mid u)$ and $\widehat{P}(A \geq 1 - \epsilon \mid u)$ for every candidate decision u, before the proposal at u^+ is generated. To test whether these classes can be predicted in advance, we train one binary classifier for each class of Section 3.3, certain accept ($A \geq 1 - \epsilon$), certain reject ($A \leq \epsilon$), and uncertain ($\epsilon < A < 1 - \epsilon$), each against the other two, using only information available before generation.

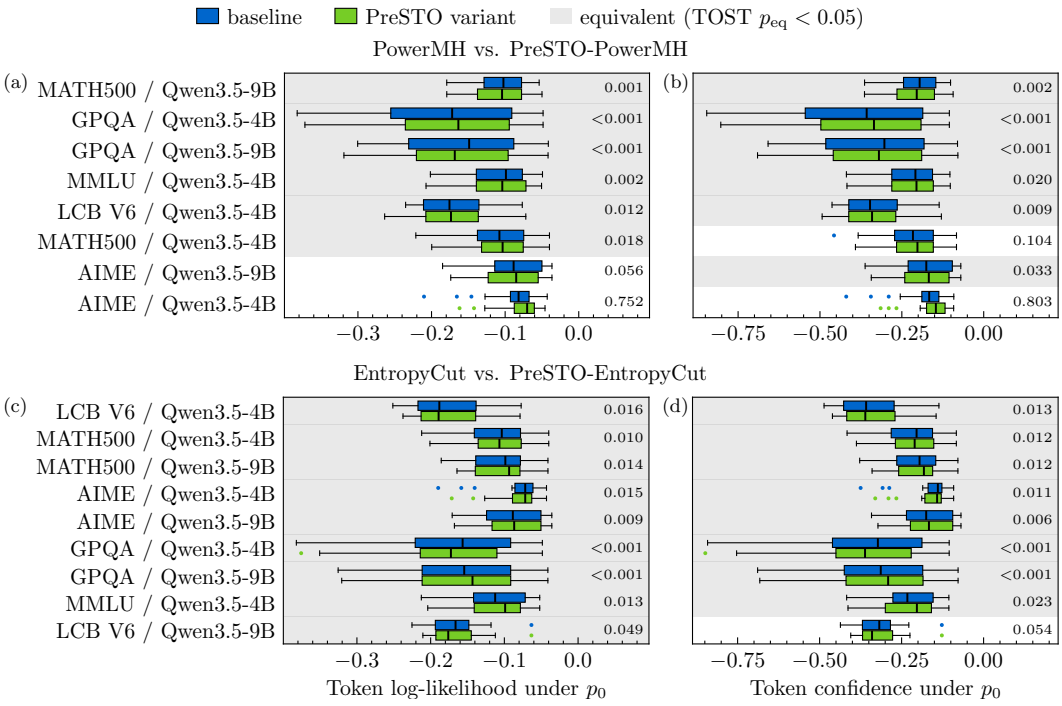


Figure 14: **Likelihood and confidence distributions with and without PreSTO.** Boxes show the per-prompt token-normalized log-likelihood (a, c) and confidence (b, d) of terminal traces under p_0 for the eight PowerMH pairs (top) and the nine EntropyCut pairs (bottom), each against its PreSTO variant; PreSTO-PowerMH and PreSTO-EntropyCut use $N_{\text{pf}} = 20$, except $N_{\text{pf}} = 10$ for PreSTO-PowerMH on LCB V6. Shaded rows are equivalent under the paired TOST ($p_{\text{eq}} < 0.05$), with p_{eq} printed at right; rows are ordered by the number of equivalent statistics.

Settings. We run PreSTO-PowerMH (uniform cuts) and PreSTO-EntropyCut (entropy threshold 4.0) with the default BFS traversal on LCB V6 with Qwen3.5-9B, with $N_{\text{pf}} = 10$, $\alpha = 4$, $K = 100$, a single block of $T = 1{,}024$ tokens, and $\epsilon = 0.01$. Each run records the acceptance probability $A = A(u^+, u)$ of every prefetched decision, and these decisions form the examples. The PreSTO-PowerMH run covers the first 20 prompts, with 579 model calls and 4,621 decisions. The PreSTO-EntropyCut run covers the first 14 prompts, with 393 model calls and 3,131 decisions. All features are available when the batch is assembled:

- *Cut and tree position:* the cut fraction $c(u^+)/T$ of the request, the cut $c(u)$ of the held state, the gap $c(u) - c(u^+)$, the depth of u, and the number of acceptances on the path from the root to u.
- *Likelihood:* the log-likelihood of the first resampled token, its changes into and out of that token, and the mean log-likelihood of the root suffix after the cut. All are computed from the token-wise log-likelihood of the root state under the proposal LLM, which applies to every decision because each state agrees with the root on positions $0, \ldots, c(u)$ (Appendix C.3).
- *Entropy* (PreSTO-EntropyCut only): the same four statistics computed from the token-wise entropy of the root state, namely the entropy at the first resampled token, its changes into and out of that token, and the mean entropy of the root suffix after the cut. The uniform-cut run did not log token entropies.

Each classifier is a logistic regression with an L_2 penalty of 1 on standardized features, a simple linear baseline for the learned predictor of Appendix C.3. We evaluate it by 5-fold cross-validation with whole prompts held out, and select features greedily by held-out log loss. We also score each feature alone by its Spearman correlation ρ with A and by its AUC as a one-feature classifier. Because the examples are the decisions that BFS actually prefetched, they follow the prefetched-tree distribution rather than the distribution of all possible decisions.

Results and analysis. For both samplers, the cut position alone is the best feature set for all three classifiers, with one small exception noted below (Table 9).

Table 9: Predicting the ϵ-certainty class before generation on LCB V6 with Qwen3.5-9B ($N_{\mathrm{pf}} = 10$, $\epsilon = 0.01$; 20 prompts for PreSTO-PowerMH, 14 for PreSTO-EntropyCut). ρ: Spearman correlation with A. AUC: each class against the other two, using $\max(\mathrm{AUC}, 1 - \mathrm{AUC})$ for single features and held-out prompts for classifiers. Bold: best per column within each block. –: entropy features are unavailable for PreSTO-PowerMH.

	PreSTO-PowerMH				PreSTO-EntropyCut			
		AUC				AUC		
Features	ρ	Certain accept vs. rest	Certain reject vs. rest	Uncertain vs. rest	ρ	Certain accept vs. rest	Certain reject vs. rest	Uncertain vs. rest
Single feature								
Cut fraction $c(u^+)/T$	**+0.51**	**0.69**	**0.79**	**0.75**	**+0.43**	**0.65**	**0.72**	**0.69**
Mean log-likelihood of root suffix	+0.32	0.60	0.65	0.63	+0.15	0.51	0.56	0.59
Log-likelihood of first resampled token	+0.07	0.54	0.54	0.52	+0.05	0.51	0.51	0.53
Mean entropy of root suffix	–	–	–	–	−0.16	0.51	0.56	0.59
Entropy at first resampled token	–	–	–	–	−0.06	0.51	0.52	0.54
Logistic classifier								
Cut and tree position	–	**0.68**	**0.77**	0.73	–	**0.63**	**0.71**	0.68
Cut, tree position, and likelihood	–	0.66	**0.77**	**0.74**	–	0.62	**0.71**	**0.70**
Cut, tree position, likelihood, and entropy	–	–	–	–	–	0.62	0.70	0.68

- *The cut is the strongest feature.* Forward selection picks it first for every classifier under both samplers. Under PreSTO-PowerMH, it has $\rho = +0.51$ with A, and it alone reaches held-out AUCs of 0.68, 0.77, and 0.73, matching the full cut-and-tree-position set. Under PreSTO-EntropyCut, it has $\rho = +0.43$, and the cut-and-tree-position set reaches 0.63, 0.71, and 0.68, within 0.01 of the cut alone.
- *Why the cut predicts acceptance.* An early cut resamples a long suffix, which is rarely accepted: under PreSTO-PowerMH, the certain-reject rate falls from 85% for cuts in the first 10% of the sequence to 3% for cuts in the last 10%. The cut is less predictive under PreSTO-EntropyCut because EntropyCut places fewer cuts near the ends of the sequence, where acceptance is most predictable: 16.5% of its decisions have their cut in the first or last 10%, against 21.3% for PreSTO-PowerMH on the same 14 prompts.
- *Other position features add nothing.* The held cut, the gap, the depth, and the number of acceptances add nothing beyond the cut.
- *Likelihood features are weak or redundant.* The first-resampled-token log-likelihood and its changes are near chance even alone ($|\rho| \leq 0.07$). The mean suffix log-likelihood is informative alone ($\rho = +0.32$ under PreSTO-PowerMH) but largely redundant given the cut. It helps in two cases: under PreSTO-EntropyCut, it raises the uncertain AUC from 0.68 to 0.70, the exception above; and under PreSTO-PowerMH on the 14 prompts shared with the PreSTO-EntropyCut run, it raises the certain-reject AUC from 0.78 to 0.80.
- *Entropy features are not useful.* The entropy at the first resampled token and its changes are near chance ($|\rho| \leq 0.06$), and the mean suffix entropy is weak ($\rho = -0.16$). Adding them lowers the held-out AUC, from 0.71 to 0.70 for certain reject and from 0.70 to 0.68 for uncertain.

The predictive traversal rule can therefore use a predictor built from the cut position alone, for both samplers, without entropy features.

E.5 EDGE CERTAINTY ACROSS MODELS AND DATASETS

This subsection extends the certainty analysis of Section 3.3 to more base LLMs, more datasets, and the EntropyCut proposal rule. Following Section 3.3 with $\epsilon = 0.01$, an MH decision with acceptance probability A is an ϵ-certain reject if $A \leq \epsilon$, an ϵ-certain accept if $A \geq 1 - \epsilon$, and ϵ-uncertain otherwise. The histograms count both outgoing edges of every scored decision, with transition probabilities A and $1 - A$, and call an edge ϵ-certain when its transition probability is at most ϵ or at least $1-\epsilon$. Because each decision contributes both A and $1-A$, the two outer categories have equal mass by construction, and their combined share equals the fraction of ϵ-certain decisions. All runs use the default BFS traversal with $N_{\text{pf}} = 20$.

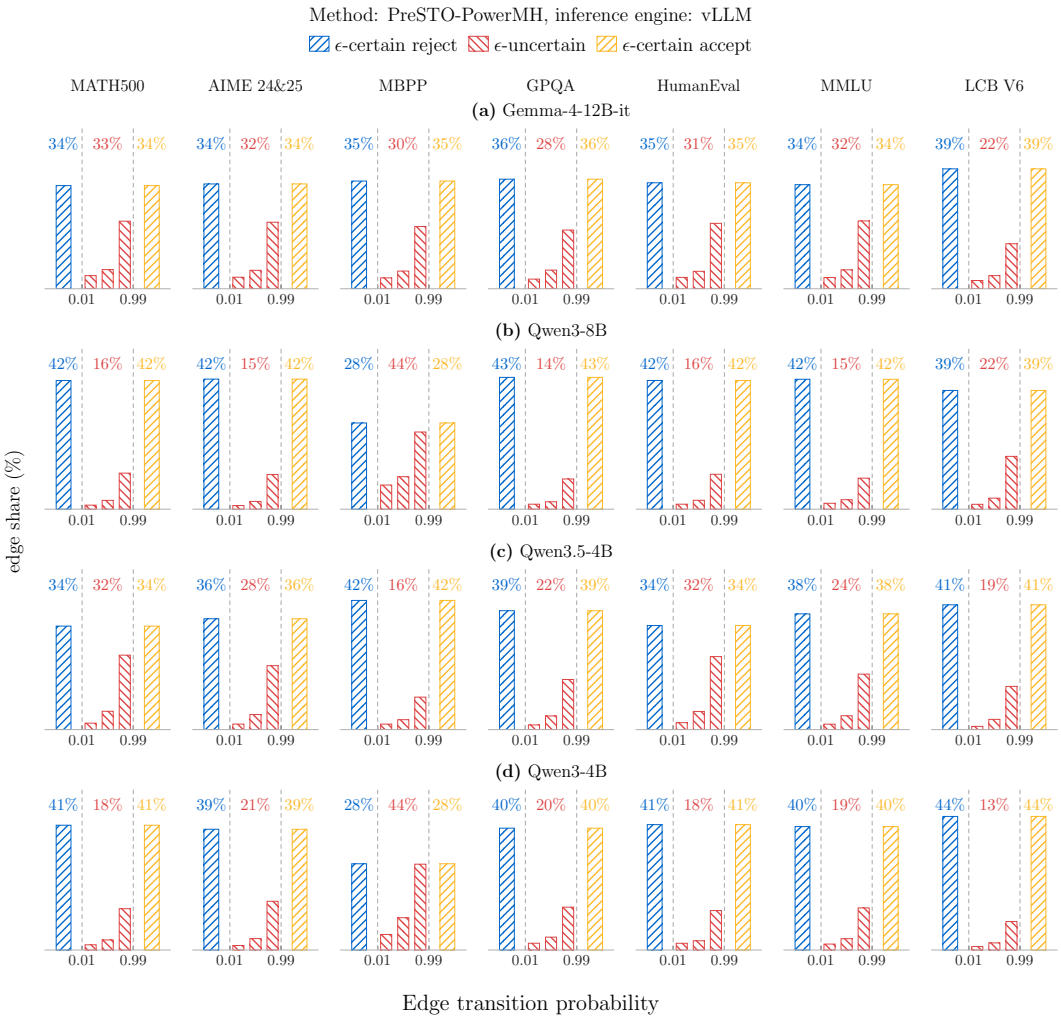


Figure 15: **Edge certainty under PreSTO-PowerMH across models and datasets.** Histograms pool acceptance and rejection edges under the default BFS traversal at $N_{\text{pf}} = 20$ and $\epsilon = 0.01$. The legend's reject/accept labels denote edges that are almost never/almost always taken, respectively, regardless of branch type. Percentages give the fraction of all scored tree edges in each probability category.

Results. Figure 15 reports PreSTO-PowerMH for four base LLMs on seven datasets, with 20 examples per dataset, $\alpha = 4$, $K = 100$, and a single block of 1,024 tokens. Figure 16 reports PreSTO-EntropyCut, which selects cut indices by token entropy (threshold 4.0), for four Qwen base LLMs on the same seven datasets. The two samplers show the same pattern. Near-deterministic decisions predominate: 56%–87% of edges are ϵ-certain for the Qwen models under PreSTO-PowerMH, and 60%–92% under PreSTO-EntropyCut.

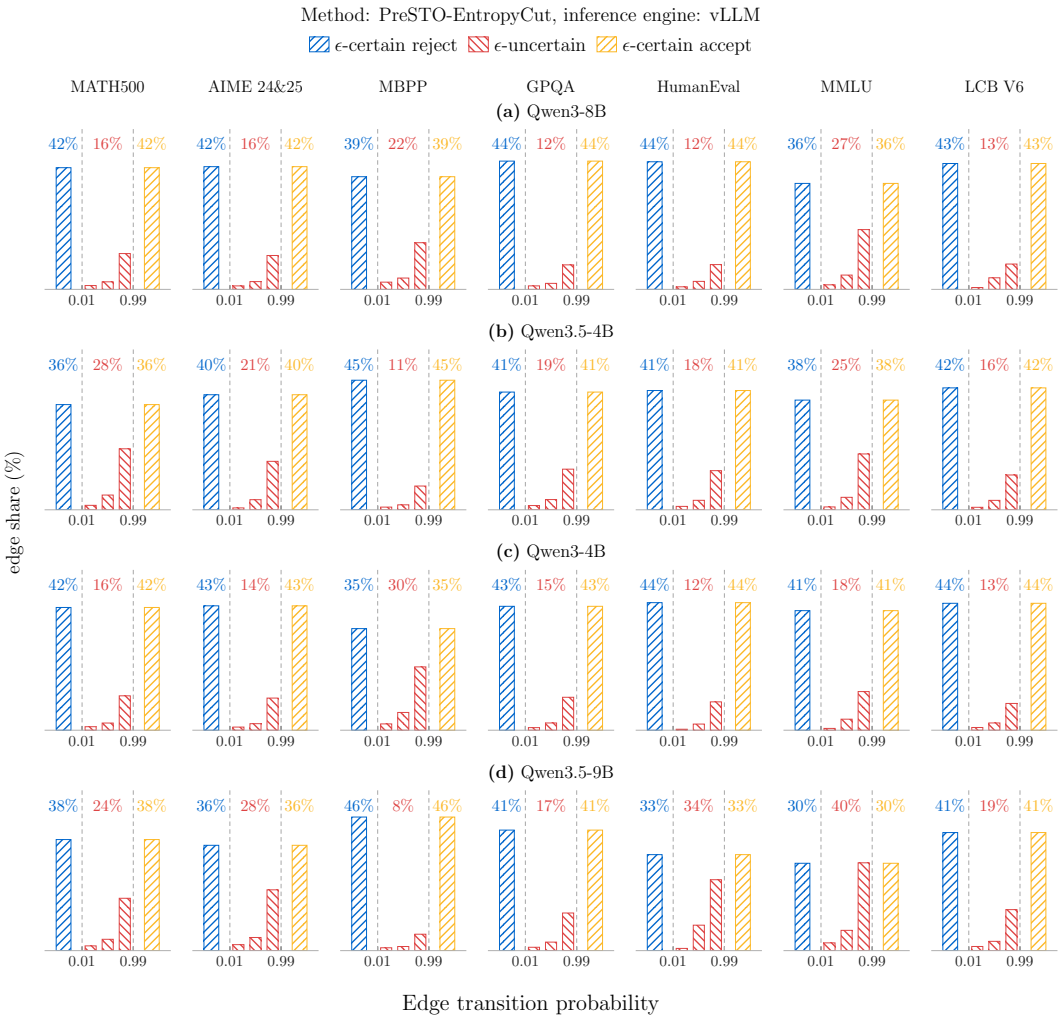


Figure 16: **Edge certainty under PreSTO-EntropyCut across models and datasets.** Histograms pool acceptance and rejection edges under the default BFS traversal at $N_{\text{pf}} = 20$ and $\epsilon = 0.01$. Percentages give the fraction of all scored tree edges in each probability category.

Comparison and scope. The comparison covers the 24 model–dataset settings excluding LCB; it uses Figure 3 for Qwen3.5-9B under PreSTO-PowerMH. PreSTO-EntropyCut has a certain-edge share at least as high as PreSTO-PowerMH in 18 of them. The largest increases are for Qwen3-8B on MBPP ($56\% \rightarrow 78\%$), Qwen3-4B on MBPP ($56\% \rightarrow 70\%$), and Qwen3.5-4B on HumanEval ($68\% \rightarrow 82\%$). Four of the six decreases are one to two percentage points. The two larger ones are both on MMLU, for Qwen3.5-9B ($83\% \rightarrow 60\%$) and Qwen3-8B ($85\% \rightarrow 73\%$). Entropy-guided cuts therefore do not remove the sparsity that PreSTO exploits, and in most settings they strengthen it. These histograms describe all scored decisions in the prefetched trees; they do not measure realized chain acceptance rates, establish the causes of runtime differences, or verify that either integration preserves the MH kernel.

E.6 Example prefetched-subtree trajectory

Execution logs of the trees. We first print the raw execution log of a single prefetched subtree: the first of the 23 subtrees from a 100-step Qwen3-4B MATH500 run with prefetch budget $N_{\text{pf}} = 18$. Box 1 reproduces the dataset prompt at index 11 verbatim. Box 2 shows the logged prefetched tree, with node IDs, cut indices, and edge labels as recorded. Its sampled path (Box 3) advances the MH chain from step 0 to step 4. Box 4 shows the ϵ-rounded tree at $\epsilon = 0.01$ (Definition 1), which omits branches assigned zero probability by rounding, together with their descendants, and retains both branches at uncertain decisions. Here every decision is certain, so the rounded tree reduces to the single path $0 \rightarrow 1 \rightarrow 3 \rightarrow 7 \rightarrow 15$, which coincides with the sampled path.

1: Problem prompt

```
Let $p(x)$ be a polynomial of degree 5 such that
\[p(n) = \frac{n}{n^2 - 1}\]for $n = 2,$ 3, 4, $\dots,$ 7.  Find $p(8).$
```

2. Prefetched tree

```
Node(id=0, cut_idx=1024)
├── [accept, log_acc_prob=0.0000] Node(id=1, cut_idx=736)
│   ├── [accept, log_acc_prob=6.7657] Node(id=3, cut_idx=628)
│   │   ├── [accept, log_acc_prob=56.8792] Node(id=7, cut_idx=330)
│   │   │   ├── [accept, log_acc_prob=51.5113] Node(id=15, cut_idx=235)
│   │   │   └── [reject] Node(id=16, cut_idx=330)
│   │   └── [reject] Node(id=8, cut_idx=628)
│   └── [reject] Node(id=4, cut_idx=736)
└── [reject] Node(id=2, cut_idx=1024)
    ├── [accept, log_acc_prob=61.6341] Node(id=5, cut_idx=746)
    │   ├── [accept, log_acc_prob=31.0659] Node(id=11, cut_idx=128)
    │   │   ├── [accept, log_acc_prob=179.3056] Node(id=23, cut_idx=107)
    │   │   └── [reject] Node(id=24, cut_idx=128)
    │   │       ├── [accept, log_acc_prob=99.3485] Node(id=49, cut_idx=90)
    │   │       └── [reject] Node(id=50, cut_idx=128)
    │   └── [reject] Node(id=12, cut_idx=746)
    │       ├── [accept, log_acc_prob=-27.4006] Node(id=25, cut_idx=114)
    │       └── [reject] Node(id=26, cut_idx=746)
    │           ├── [accept, log_acc_prob=68.1415] Node(id=53, cut_idx=331)
    │           │   ├── [accept, log_acc_prob=71.1094] Node(id=107, cut_idx=4)
    │           │   └── [reject] Node(id=108, cut_idx=331)
    │           └── [reject] Node(id=54, cut_idx=746)
    │               ├── [accept, log_acc_prob=13.6195] Node(id=109, cut_idx=501)
    │               └── [reject] Node(id=110, cut_idx=746)
    └── [reject] Node(id=6, cut_idx=1024)
        ├── [accept, log_acc_prob=29.2494] Node(id=13, cut_idx=296)
        └── [reject] Node(id=14, cut_idx=1024)
            ├── [accept, log_acc_prob=-45.6767] Node(id=29, cut_idx=760)
            └── [reject] Node(id=30, cut_idx=1024)
                ├── [accept, log_acc_prob=63.0602] Node(id=61, cut_idx=283)
                └── [reject] Node(id=62, cut_idx=1024)
```

3. MH progress 0/100 -> 4/100 (4 transitions)

```
sampled path in the prefetched subtree: 0 -> 1 -> 3 -> 7 -> 15
```

4. ϵ-rounded tree ($\epsilon = 0.01$)

```
Node(id=0, cut_idx=1024)
└── [accept, log_acc_prob=0.0000] Node(id=1, cut_idx=736)
    └── [accept, log_acc_prob=6.7657] Node(id=3, cut_idx=628)
        └── [accept, log_acc_prob=56.8792] Node(id=7, cut_idx=330)
            └── [accept, log_acc_prob=51.5113] Node(id=15, cut_idx=235)
```

Binary-tree views of prefetched subtrees. We plot all 21 prefetched subtrees for prompt index 0 from the Qwen3.5-9B run on LCB V5 with prefetch budget $N_{\text{pf}} = 20$. Figure 17 shows them in the logged order; together they cover the chain's full trajectory of 100 MH transitions. Each panel overlays three views of one subtree: the full prefetched tree, its ϵ-rounded subtree at $\epsilon = 0.01$ (Definition 1), and the sampled path. The rounded subtree keeps only the nodes reachable from the root after rounding, and keeps both branches at decisions that remain uncertain. The figure caption gives the color and edge conventions.

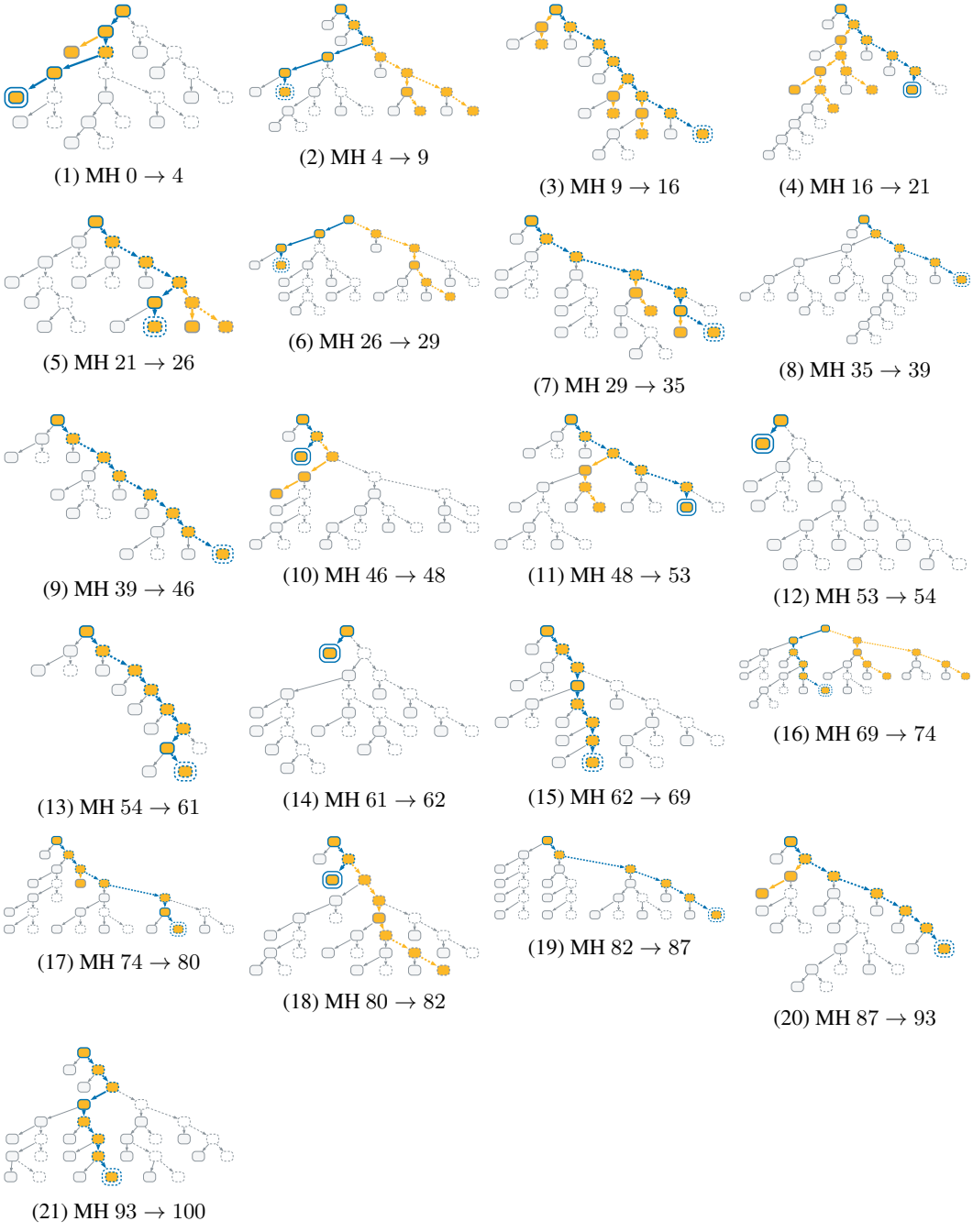


Figure 17: **Prefetched subtrees for the full trajectory of 100 MH transitions.** All 21 subtrees in sampling order; each label gives the MH steps that the subtree advances. Solid left edges denote acceptance and dashed right edges rejection. Orange nodes and edges mark the ϵ-rounded subtree at $\epsilon = 0.01$ (Definition 1), blue outlines and thick blue edges mark the sampled path (terminal node double-bordered), and gray marks the remaining prefetched branches.