Sampling from Energy-based Language Models with Multiple Tries

TJ Bai
Johns Hopkins University
Baltimore, MD
tbai4@jhu.edu

Abstract

Standard MCMC sampling for large language models (LLMs) typically suffers from long mixing times. We explore Multiple-try Metropolis (MTM), which draws *multiple* proposals per step to accelerate mixing while leveraging GPU parallelism. On a toy model and GPT-2, MTM sometimes converges faster than vanilla Metropolis–Hastings, but the benefit depends sensitively on the hyper-parameters and can be offset by higher compute cost. These nuanced findings suggest MTM is a promising, though not yet turnkey, ingredient for practical EBM sampling with LLMs.

1 Introduction

LLMs are powerful probabilistic models of text (Minaee et al., 2024). Yet their autoregressive nature makes certain inference problems—e.g. controlled generation (Zhang et al., 2023) or prompt inversion (Zhang et al., 2024)—hard to sample from directly.

Recasting such tasks as sampling from an appropriately defined energy-based model (EBM) enables Markov Chain Monte Carlo (MCMC). Unfortunately, generic MCMC often mixes slowly in high-dimensional language spaces. We therefore test *Multiple-try Metropolis* (MTM; Liu et al., 2000), which proposes a batch of k states per iteration. Larger batches exploit GPU parallelism and, in principle, shorten the path to equilibrium.

Across a toy Ising model and GPT-2 (§4) we find MTM *can* improve per-step mixing given carefully tuned step sizes and batch sizes. However, the extra within-step compute means wall-clock speed-ups are far from guaranteed (Figure 2, Figure 5). Even our best variant still costs $1.8\times$ the time of vanilla MCMC, indicating that multiple tries are promising but not yet a drop-in replacement.

2 Primer

2.1 Language Models

Traditionally, Transformer-based LLMs factorize a distribution over strings $p_{\text{LM}}(x_{1:n})$ as a product of locally-normalized conditional distributions.

$$p_{\text{LM}}(x_{1:n}) = p_{\text{LM}}(x_1) \prod_{i=2}^{n} p_{\text{LM}}(x_i \mid x_{1:i-1})$$

This formulation enables efficient training on massive text corpora and has nice properties at inference-time. Notably, we can efficiently sample strings from the LLM from left-to-right by sequentially sampling each token conditioned on its predecessors, a form of *ancestral sampling*.

However, there are many applications where we may want to *incorporate* an LLM probability, but modify the distribution such that autoregressive sampling is not possible.

For instance, in *controlled generation*, we may define arbitrary soft constraints on the text, such as the likelihood under a probabilistic context-free-grammar or to satisfy some preference model. Such constraints are defined on the *complete* string, so in general we can not sample faithfully from the target distribution autoregressively. Another application is *prompt inversion*, where we may seek to sample a string $x_{1:i}$ that is likely to generate an observed sequence $x_{i+1:n}$. Sampling from the exact posterior distribution $p(x_{1:i} \mid x_{i+1:n})$ involves an intractable denominator over $|\Sigma|^i$ strings for generally very large $|\Sigma|$.[1]

2.2 Energy-based Models

We can model a large class of inference problems that incorporate a language model probability with

[1] Σ denotes the *vocabulary* of the language model, where each string belongs to the set Σ^*. For reference, the popular consumer LLM GPT-4o has a vocabulary size close to 200,000.

an intractable denominator as sampling from an EBM.

$$\pi(x) = \frac{1}{Z} \exp\{-\beta E(x)\}$$

We denote $E : \Omega \to \mathbb{R}$ the *energy function*, which assigns a score to each item in the state space such that *high* energy values have *low* probability, and vice versa. Z is a normalization factor, commonly called the *partition function*, that ensures the distribution has unit measure, though we often cannot compute this tractably.

This formalism has appealing flexibility, as we can define a target distribution solely in terms of an appropriate energy function. For instance, $E(x) = -\log p_{\text{LLM}}(x)$ corresponds to a baseline LLM distribution with inverse temperature scaling β. Incorporating additional terms then allows us to modify the target distribution for the problem at hand.

1. Controlled Generation:
   $\pi(x) \propto p_{\text{LLM}}(x) \prod_{i=1}^{k} \log p_{\text{CLS}_i}(x)$
   $E(x) = -\log p_{\text{LLM}}(x) - \sum_{i=1}^{k} \log p_{\text{CLS}_i}(x)$
2. Prompt Inversion:
   $\pi(x) \propto p_{\text{LM}}(x_{1:i}) p_{\text{LM}}(x_{j+1:n} \mid x_{1:i})$
   $E(x) = -\log p_{\text{LM}}(x_{1:n})$[2]

2.3 Markov Chain Monte Carlo

To sample from an EBM, we can turn to the classic MCMC algorithm. For simplicity, consider a state space of *fixed-length* strings in Σ^n. Starting from an initial (possibly randomly-initialized) string x_0, in each MCMC iteration we:

1. Sample a new state from a *proposal distribution* conditioned on the current state: $q(x' \mid x_i)$.
2. Apply the *Metropolis-Hastings correction* to compute acceptance ratio $A = \frac{\pi(x')q(x_i|x')}{\pi(x_i)q(x'|x_i)}$.
3. Set $x_{i+1} := x'$ with probability $\min(1, A)$, otherwise $x_{i+1} := x_i$.

MCMC iteratively constructs a Markov chain of samples from the state space. The Metropolis-Hastings (MH) correction attempts to encourage samples that are more likely under the target distribution π, while correcting for any sampling bias in our proposal distribution q. Importantly, this entire procedure is tractable—we sample from a hand-designed proposal function, q, and $\frac{\pi(x')}{\pi(x_i)}$ simplifies to $\frac{\exp\{-\beta E(x')\}}{\exp\{-\beta E(x_i)\}}$, allowing us to avoid computing the intractable partition function.

Rigorously, it is straightforward to show that the MH procedure guarantees that our Markov chain of states is reversible with respect to π, because $\pi(x_i)P(x_i \to x_j) = \pi(x_j)P(x_j \to x_i)$, where $P(x_i \to x_j) = q(x_j \mid x_i) \times \min(1, A)$. This implies that π is a *stationary distribution* of the Markov chain. If we choose q such that the chain is also *irreducible*, implying that every state is reachable after a sufficient number of steps, then this stationary distribution is actually the unique *limiting distribution* of the Markov chain.

Theorem 1 (Fundamental Limit Theorem for Ergodic Markov Chains)**.** *Let $X_0, X_1, \ldots$ be an ergodic Markov chain. There exists a unique, positive, stationary distribution π, which is the limiting distribution of the chain. That is, $\pi_j = \lim_{n\to\infty} P_{ij}^n$ for all i, j.*

In practice, this means that running MCMC for a large enough number of iterations guarantees that our Markov chain will eventually converge to the target distribution. This makes it an appealing general-purpose sampling algorithm when exact methods are not available.

Of course, this method does not come without its downsides. Denoting the empirical distribution over states at step n with initial state x as P_x^t, the *ϵ-mixing time* is defined as:

$$t_{\text{mix}}(\epsilon) = \inf\left\{t : \sup_{x \in \Sigma^n} \left\|P_x^t, \pi\right\|_{\text{TV}} \leq \epsilon\right\}$$

This is the smallest number of steps required for the chain's distribution to come within ϵ of the target distribution in total variation distance (TVD), regardless of the initial state. In most high-dimensional problems, such as sampling strings of length n over an alphabet Σ, the mixing time grows at least linearly (and often worse) in the number of coordinates. Since the total state space size is $|\Sigma|^n$, naive MCMC may require impractically many steps to mix well in practice.

[2]This simplifies because $p_{\text{LM}}(x_{1:i})p_{\text{LM}}(x_{i+1:n} \mid x_{1:i})$. However, this is not equivalent to unconditional sampling because the tokens $x_{i+1:n}$ are *fixed*.

3 Multiple-try Metropolis

We build on the Multiple-try Metropolis (MTM) scheme of Liu et al. (2000), which generalizes Metropolis–Hastings by generating a batch of k proposals at each step and then selecting among them. Concretely, given current state x_i:

1. Draw k candidates $y_1, \dots, y_k \sim q(\cdot \mid x_i)$.
2. For each $j = 1, \dots, k$, set $w_j = \pi(y_j)q(x_i \mid y_j)\lambda(x_i, y_j)$ for any symmetric function λ.
3. Sample y^* from $\{y_1, \dots, y_k\}$ with probability proportional to w_j.
4. Draw $k-1$ reverse points $x'_1, \dots, x'_{k-1}$ from $q(\cdot \mid y^*)$ and set $x'_k = x_i$. Compute the corresponding reverse weights w'_i.
5. Transition $x_i \to y^*$ with probability

$$\alpha = \min\left(1, \frac{\sum_{i=1}^k w_i}{\sum_{i=1}^k w'_i}\right),$$

otherwise set $x_{i+1} := x_i$.

Notice that each MTM step performs $2k$ energy function evaluations versus just one in vanilla MH. However, by exploring k points per iteration, we hope to increase the probability of finding a satisfactory point to transition to, potentially reducing mixing time in hard-to-navigate energy landscapes.

Additionally, because our energy function is typically evaluated by a Transformer LLM and other neural scoring functions, MTM evaluations can easily take advantage of GPU parallelism. Thus, the per-step overhead for multiple evaluations scales *sublinearly* with respect to batch size, which we hope to offset by reducing the number of *total* MCMC iterations required to mix well.

For the proposal distribution, we adopt p-NCG introduced by Du et al. (2024), which leverages energy gradients on an *embedding*[3] of tokens in the string to suggest moves. The required gradients can be efficiently computed by performing *back-propagation* after we evaluate the energy function in the forward pass.

$$q(x' \mid x) \propto \exp\Big\{ -\frac{1}{2}\nabla E(x)^\top (x' - x) - \frac{1}{2\alpha}\|x' - x\|_p^p \Big\}$$

[3]An embedding is a vector in $\mathbb{R}^d$ representing each token. Specifically, these embeddings are the layer 0 non-contextual representations of each token, which is required for efficiency's sake.

Additionally, the MTM framework allows for an arbitrary symmetric function $\lambda(x, y)$ to be chosen, where different choices may plausibly have a large impact on mixing time. Aside from a baseline MTM method where $\lambda(x, y) = 1$, we also experiment with the "importance-weighted" variant (IW-MTM) where $\lambda(x, y) = \big(q(x \mid y)q(y \mid x)\big)^{-1}$. The name is derived from observing that this sets $w_j = \pi(y_j)q(y_j \mid x_i)^{-1}$, which corresponds to the importance weight if our target distribution is π and proposal distribution is q. This method also allows us to cache the proposal distribution when computing the MTM weights, reducing the total number of forward-backward passes and improving efficiency. For further implementation details, we refer the reader to `https://github.com/tjbai/mtmcmc/`.

4 Experiments

4.1 Toy Language Model

Following (Du et al., 2024), we first conduct experiments in a toy setting where we can compute the target distribution exactly. We design a toy energy-based language model which corresponds to a linear-chain Ising model with energy function

$$E(x) = \frac{1}{2}x^\top Ax$$

where A is the adjacency matrix of an n-cycle and $x \in \{-1, 1\}^n$. Computing the exact distribution requires evaluating the energy for 2^n total states, which is tractable for relatively small n. Note that because of the cyclical dependency between "tokens" in this setting, we cannot perform ancestral sampling, motivating MCMC as a natural approximate sampling method.

In Figure 4, we evaluate the TVD between the empirical and exact distributions after 1000 MCMC iterations across various step sizes, α, and batch sizes, k. Taking the best-performing hyperparameters for each method, we evaluate their performance after 1000 steps (Figure 3) and compare the convergence over time (Figure 1).

5 "Large" Language Model

We apply our method to GPT-2, a seminal 124M parameter LLM (Radford et al., 2019).[4] With a string length $n = 20$, we set our target distribution to the *unconstrained* LLM distribution, e.g. $E(x) = -\log p_{\text{LLM}}(x)$. Because computing TVD is intractable, we evaluate the reduction in energy

[4]This is *very* small by modern standards.

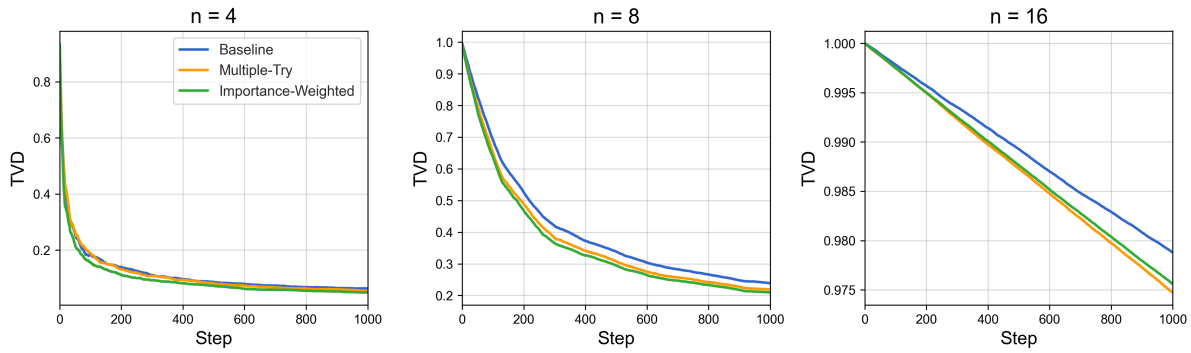

Figure 1: TVD over time for each MCMC method sampling from an Ising model with dimension n. Optimal hyperparameters obtained via grid search. Note from the y-axis that convergence becomes significantly more difficult as dimensionality increases. In each setting, we observe small but consistent improvements compared to the baseline. However, these results do not appear to be *significantly* better (Figure 3) and come at non-trivial additional cost.

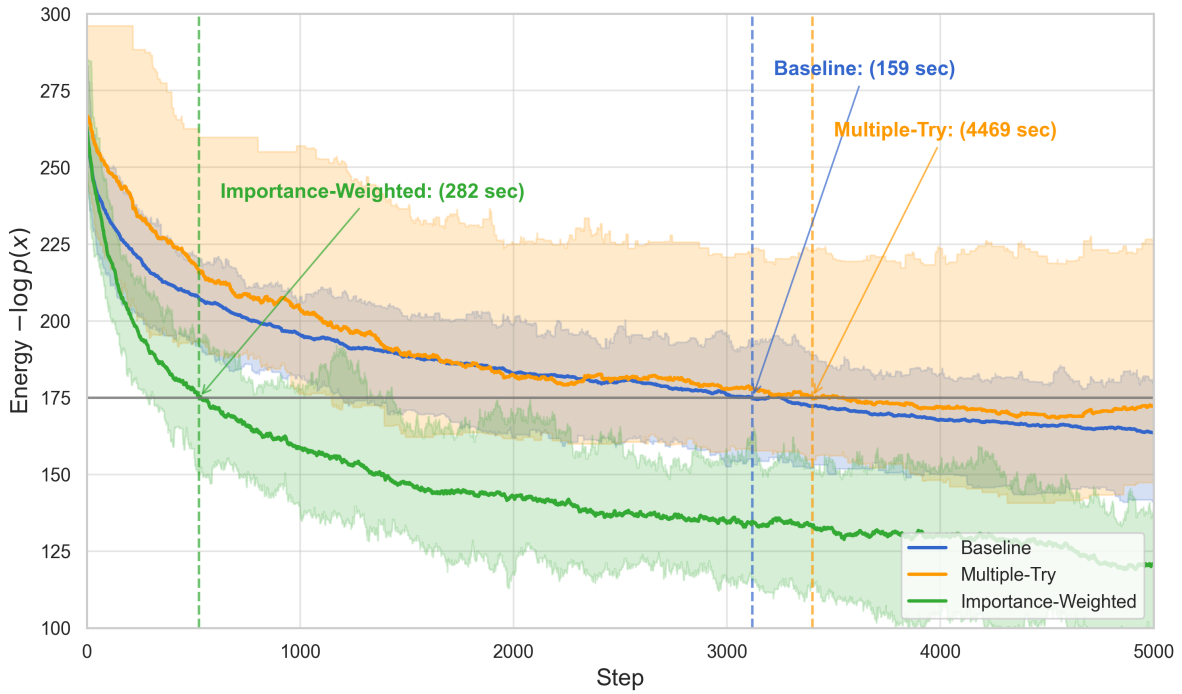

Figure 2: We plot mean energy over 5,000 MCMC iterations averaged across 50 independent runs. Filled in areas represent 95% empirical confidence intervals. The grey horizontal bar marks an energy of 175 and dashed vertical lines indicate the first step where each method's mean trajectory reaches that threshold. Labels represent wallclock time required to reach that threshold. While naive MTM does not appear to improve over the baseline, and seems to have much higher inter-run variance, IW-MTM is more sample efficient and reaches the threshold in just over 500 steps versus more than 3,000 in the baseline. Despite this, the wallclock time required to reach that threshold is still $\approx 1.8\times$ longer because of the greater within-step compute cost. See Figure 5 for details.

Method	n	α	k	TVD	Time (s)
Baseline	4	64.0	1	0.057 ± 0.011	1.54
MTM	4	28.9	16	0.055 ± 0.014	2.50
IW-MTM	4	28.9	16	0.048 ± 0.009	1.97
Baseline	8	01.2	1	0.239 ± 0.013	1.63
MTM	8	64.0	32	0.222 ± 0.008	2.56
IW-MTM	8	64.0	32	0.207 ± 0.010	2.09
Baseline	16	00.5	1	0.978 ± 0.001	1.84
MTM	16	28.9	32	0.975 ± 0.001	2.85
IW-MTM	16	28.9	32	0.975 ± 0.001	2.29

Figure 3: Comparison of MCMC sampling from Ising model with optimal step and batch sizes obtained from grid search. We perform 30 trials and compute 95% confidence intervals via bootstrapping.

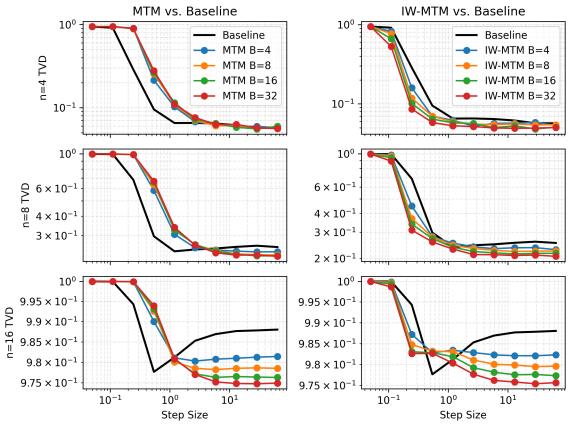

Figure 4: Ising model TVD after 1000 steps. For specific settings of the step and batch size, both MTM variants reach lower TVD than the baseline.

over a fixed number of steps as a proxy measure of convergence.[5]

To determine the optimal hyperparameter setting, we sweep α at evenly-spaced log-intervals in $(0.1, 64)$ and $k \in \{4, 8, 16, 32, 64, 128, 256, 512\}$. As grid search is too expensive for our compute budget, we perform 250 iterations of Bayesian hyperparameter optimization and HyperBand (Li et al., 2017). We find that larger batch sizes and smaller step sizes generally help in Figure 6. Using the best hyperparameters for each method, we compare and discuss convergence rates in Figure 2. All experiments are conducted on a single A100-40GB GPU.

[5]Note that the empirical distribution allows us to compute a Monte Carlo estimate of $H(P_x^t, \pi)$, the cross-entropy between empirical and target distributions, which only differs from the KL divergence by a constant.

Method	Energy 95% CI	Wallclock (s) 95% CI
Baseline	163.7 (158.9, 169.8)	255 (254.4, 255.1)
MTM	172.2 (158.0, 194.0)	6570 (6529, 6627)
IW-MTM	121.0 (116.4, 125.8)	2685 (2214, 3323)

Figure 5: GPT-2 sampling results after 5000 MCMC iterations. We report 95% CIs obtained via bootstrapping over 50 runs. See Figure 2 for convergence plots.

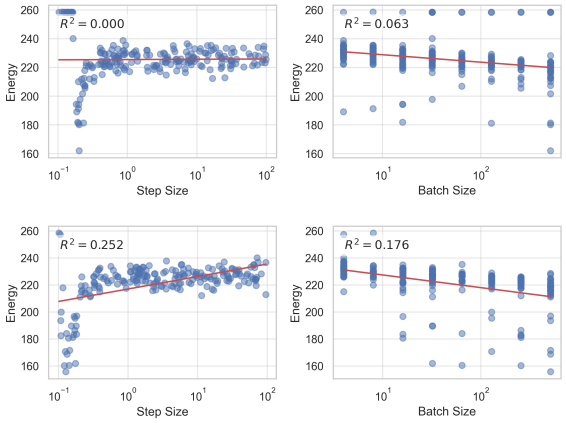

Figure 6: Results from hyperparameter sweep for GPT-2 sampling. We plot the energy after 1000 steps with various step and batch sizes. **Top** shows MTM while **bottom** shows IW-MTM. Note that the x-axis is in log-space. Increasing batch size and decreasing step size tend to help with convergence, except for some outlier failure modes in naive MTM.

6 Discussion

Summary Our results are mixed. On the toy Ising model MTM behaves exactly as theory predicts: more proposals per step lower total-variation distance with little extra time. For GPT-2, the importance-weighted variant (IW-MTM) reaches the same energy basin in roughly one-seventh the *iterations* of vanilla Metropolis–Hastings, yet still needs about $1.8\times$ the wall-clock time because each step is heavier.

Limitations We study only short sequences ($n = 20$), a single gradient-based proposal family (p-NCG), and the easiest task—unconditional sampling. Future work in more diverse settings is needed.

Future work Two avenues look most promising: (i) *adaptive batching*, where the chain enlarges k only when progress stalls, and (ii) *smarter proposals*, such as learned moves, that make the energy landscape fundamentally easier to navigate.

7 Acknowledgments

Portions of this work was carried out using the Advanced Research Computing at Hopkins (ARCH) core facility (rockfish.jhu.edu), which is supported by the National Science Foundation grant number OAC1920103. We thank Jason Eisner and Li Du for helpful discussion and comments.

References

Li Du, Afra Amini, Lucas Torroba Hennigen, Xinyang Velocity Yu, Holden Lee, Jason Eisner, and Ryan Cotterell. 2024. Principled gradient-based MCMC for conditional sampling of text. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, ICML'24. JMLR.org.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2017. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research (JMLR)*, 18(1):6765–6816.

Jun S. Liu, Faming Liang, and Wing Hung Wong. 2000. The multiple-try method and local optimization in Metropolis sampling. *Journal of the American Statistical Association*, 95(449):121–134.

Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. 2024. Large language models: A survey. *Computing Research Repository*, arXiv:2402.06196.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9.

Collin Zhang, John Xavier Morris, and Vitaly Shmatikov. 2024. Extracting prompts by inverting LLM outputs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 14753–14777. Association for Computational Linguistics.

Hanqing Zhang, Haolin Song, Shaoyu Li, Ming Zhou, and Dawei Song. 2023. A survey of controllable text generation using Transformer-based pre-trained language models. *ACM Computing Surveys*, 56(3):1–37.