# Sampling from Energy-based Language Models with *Multiple-try Metropolis*

## Introduction to Stochastic Processes

**TJ Bai**

# Large Language Models (LLMs)

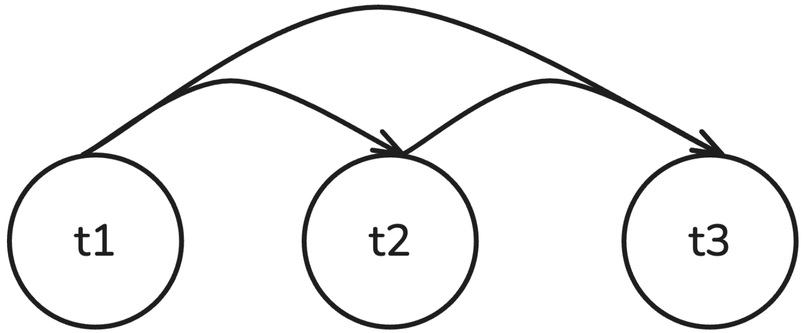
Remarkably powerful probability distributions over strings

Dominant capabilities in a wide variety of applications

# Large Language Models (LLMs)

Commonly factorize the distribution as a product of ***locally normalized*** distributions

$$p(t_{1:n}) = p(t_1)\prod_{i=2}^{n} p(t_i \mid t_{1:i-1})$$




Easy to sample outputs from left-to-right—an instance of ***ancestral sampling***

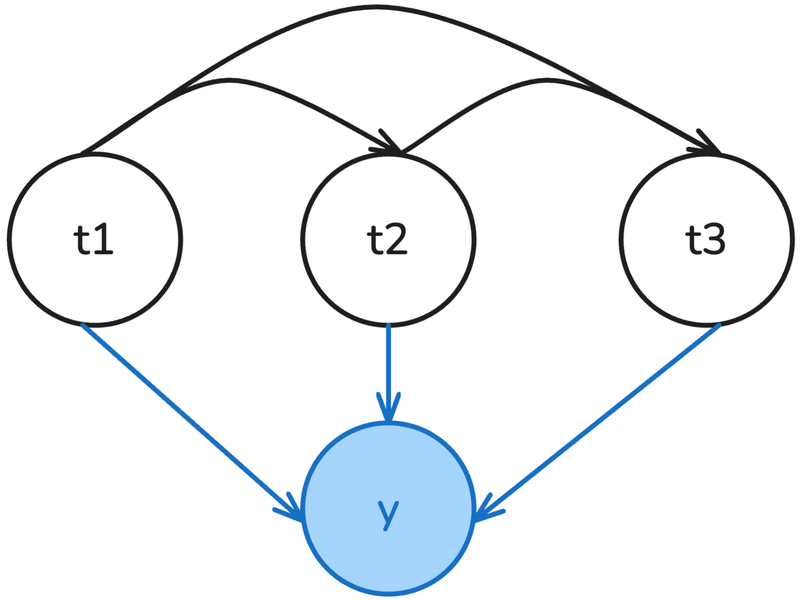
# What if we ***can't*** sample left-to-right?

# What if we can’t sample left-to-right?

## Example: Controlled Generation

We want samples that have high probability under the language model ***and some global criterion***

$$p(x) = p_{\text{LLM}}(x) \prod_i p_{\text{CLS}_i}(x)$$




Examples:

- Code that compiles under a specific grammar
- Aligning outputs to a reward/preference model

# What if we can’t sample left-to-right?

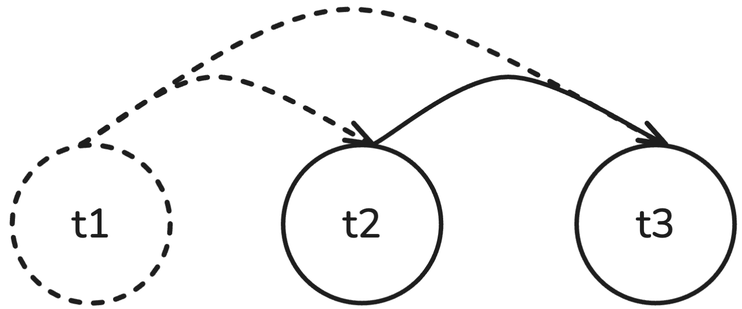
## Example: Prompt Inversion

We want to sample ***prompts*** that will elicit fixed ***outputs***

$$p(\text{prompt} \mid \text{outputs}) = \frac{p(\text{prompt})p(\text{outputs} \mid \text{prompt})}{\sum p(\text{prompt})p(\text{outputs} \mid \text{prompt})}$$

Examples:

- Evaluating robustness to adversarial users



# These are *hard* distributions

Computing the ***exact*** distribution is generally intractable

We can’t just brute-force all $|\text{vocabulary}|^N$ strings!

# These are *hard* distributions

Computing the ***exact*** distribution is generally intractable

We could do…

- Rejection sampling
- Importance sampling
- Particle filters
- Twisted particle filters
- Variational inference
- Distillation
- GFlowNets

# These are *hard* distributions

Computing the ***exact*** distribution is generally intractable.

We could do…

- Rejection sampling
- Importance sampling
- Particle filters
- Twisted particle filters
- Variational inference
- Distillation
- GFlowNets
- ***MCMC!***

# Energy-based Language Models

Energy-based models: $\pi(x) \propto e^{-\beta E(x)}$

Controlled generation: $E(x) = -\log p(x) + \sum_i \text{score}_i(x)$

Prompt inversion: $E(x) = -\log p(\text{prompt}) - \log p(\text{outputs} \mid \text{prompt})$

# MCMC for EBMs

1. Sample a new state from our proposal distribution: $q(x' \mid x)$
2. Compute the Metropolis-Hastings Correction: $A = \frac{\pi(x')q(x \mid x')}{\pi(x)q(x' \mid x)}$
3. Transition to the new state with probability $min(1, A)$

# MCMC for EBMs

1. Sample a new state from our proposal distribution: $q(x' \mid x)$
2. Compute the Metropolis-Hastings Correction: $A = \frac{\pi(x')q(x \mid x')}{\pi(x)q(x' \mid x)}$
3. Transition to the new state with probability $min(1, A)$

Denominators cancel!

$$\frac{\pi(x')}{\pi(x)} = \frac{e^{-\beta E(x')}}{e^{-\beta E(x)}}$$

# Why does this work?

Metropolis-Hastings ensures detailed balance:

$$\pi(x')p(x \to x') = \pi(x)p(x' \to x)$$

This is a sufficient condition for $\pi(x)$ to be a stationary distribution.

So, if we choose a proposal distribution $p(\,\cdot \mid x)$ that is *ergodic…*

# Why does this work?

Metropolis-Hastings ensures detailed balance:

$$\pi(x')p(x \to x') = \pi(x)p(x' \to x)$$

This is a sufficient condition for $\pi(x)$ to be a stationary distribution.

So, if we choose a proposal distribution $p(\cdot \mid x)$ that is *ergodic…*

**Fundamental Limit Theorem of Ergodic Markov Chains:**

***This stationary distribution is the unique limiting distribution!***

# Proposal Distribution

## Challenge: Language is *discrete*

We could…

- Generate a new independent sample (Independent Metropolis-Hastings)
- Re-sample portions of the string (Top-k Block Gibbs, QUEST, CGMH, *inter alia*)
- Relax the string into continuous embedding space (MuCoLA, COLD, SVS)
- Prompt another language model to propose a new string (???)
- …

# Proposal Distribution

## Challenge: Language is *discrete*

We could…

- Generate a new independent sample (Independent Metropolis-Hastings)
- Re-sample portions of the string (Top-k Block Gibbs, QUEST, CGMH, *inter alia*)
- Relax the string into continuous embedding space (MuCoLA, COLD, SVS)
- Prompt another language model to propose a new string (???)
- …
- ***p-NCG** (Du et al. 2024) — Discrete proposal with gradient-based exploration*

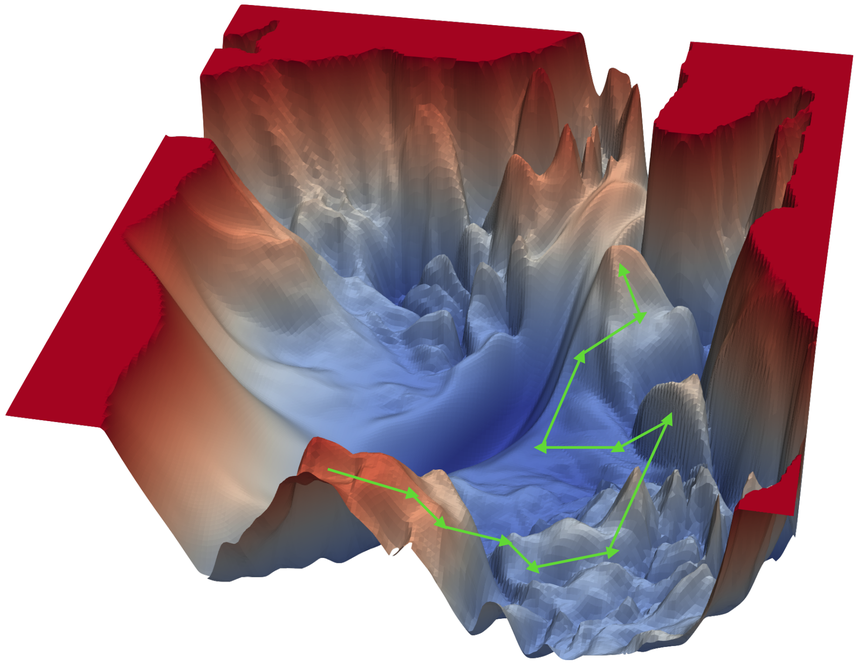
# Mixing Time

## Problem: Convergence is too slow

Navigating the energy landscape efficiently can be very difficult

If we sample long strings, curse of dimensionality strikes

Common for mixing time to be on the order of ***hours*** for even small LLMs



# Mixing Time

## Solution: Multiple-try Metropolis?

Idea: Sample ***many*** proposals at each step to accelerate mixing time

# Mixing Time

## Solution: Multiple-try Metropolis?

Idea: Sample ***many*** proposals at each step to accelerate mixing time

Liu et al. (2000) gives a principled method to do this

As in a standard M–H algorithm, we let $T(\mathbf{x}, \mathbf{y})$ be a proposal transition function that may or may not be symmetric. A modest requirement is that $T(\mathbf{x}, \mathbf{y}) > 0$ if and only if $T(\mathbf{y}, \mathbf{x}) > 0$. Furthermore, we define

$$w(\mathbf{x}, \mathbf{y}) = \pi(\mathbf{x}) T(\mathbf{x}, \mathbf{y}) \lambda(\mathbf{x}, \mathbf{y}), \tag{2}$$

where $\lambda(\mathbf{x}, \mathbf{y})$ is a nonnegative symmetric function in $\mathbf{x}$ and $\mathbf{y}$ that can be chosen by the user. The only requirement is that $\lambda(\mathbf{x}, \mathbf{y}) > 0$ whenever $T(\mathbf{x}, \mathbf{y}) > 0$. We present a few choices of $\lambda(\mathbf{x}, \mathbf{y})$ in the latter part of this section. Suppose that the *current state* is $\mathbf{X}_t = \mathbf{x}$; then a MTM transition is defined as follows.

**Multiple-try Metropolis**

1. Draw $k$ iid trial proposals, $\mathbf{y}_1, \ldots, \mathbf{y}_k$, from $T(\mathbf{x}, \cdot)$. Compute $w(\mathbf{y}_j, \mathbf{x})$ for $j = 1, \ldots, k$.
2. Select $\mathbf{Y} = \mathbf{y}$ among the trial set $\{\mathbf{y}_1, \ldots, \mathbf{y}_k\}$ with probability proportional to $w(\mathbf{y}_j, \mathbf{x}), j = 1, \ldots, k$. Then draw $\mathbf{x}_1^*, \ldots, \mathbf{x}_{k-1}^*$ from the distribution $T(\mathbf{y}, \cdot)$, and let $\mathbf{x}_k^* = \mathbf{x}$.
3. Accept $\mathbf{y}$ with probability

$$r_g = \min \left\{ 1, \frac{w(\mathbf{y}_1, \mathbf{x}) + \cdots + w(\mathbf{y}_k, \mathbf{x})}{w(\mathbf{x}_1^*, \mathbf{y}) + \cdots + w(\mathbf{x}_k^*, \mathbf{y})} \right\} \tag{3}$$

and reject it with probability $1 - r_g$. The quantity $r_g$ is called the *generalized M–H ratio.*

# Mixing Time

## Solution: Multiple-try Metropolis?

Idea: Sample ***many*** proposals at each step to accelerate mixing time

Liu et al. (2000) gives a principled method to do this

***Bonus: GPUs are really good at computing things in parallel***

As in a standard M–H algorithm, we let $T(\mathbf{x}, \mathbf{y})$ be a proposal transition function that may or may not be symmetric. A modest requirement is that $T(\mathbf{x}, \mathbf{y}) > 0$ if and only if $T(\mathbf{y}, \mathbf{x}) > 0$. Furthermore, we define

$$w(\mathbf{x}, \mathbf{y}) = \pi(\mathbf{x}) T(\mathbf{x}, \mathbf{y}) \lambda(\mathbf{x}, \mathbf{y}), \tag{2}$$

where $\lambda(\mathbf{x}, \mathbf{y})$ is a nonnegative symmetric function in $\mathbf{x}$ and $\mathbf{y}$ that can be chosen by the user. The only requirement is that $\lambda(\mathbf{x}, \mathbf{y}) > 0$ whenever $T(\mathbf{x}, \mathbf{y}) > 0$. We present a few choices of $\lambda(\mathbf{x}, \mathbf{y})$ in the latter part of this section. Suppose that the *current state* is $\mathbf{X}_t = \mathbf{x}$; then a MTM transition is defined as follows.

**Multiple-try Metropolis**

1. Draw $k$ iid trial proposals, $\mathbf{y}_1, \ldots, \mathbf{y}_k$, from $T(\mathbf{x}, \cdot)$. Compute $w(\mathbf{y}_j, \mathbf{x})$ for $j = 1, \ldots, k$.
2. Select $\mathbf{Y} = \mathbf{y}$ among the trial set $\{\mathbf{y}_1, \ldots, \mathbf{y}_k\}$ with probability proportional to $w(\mathbf{y}_j, \mathbf{x}), j = 1, \ldots, k$. Then draw $\mathbf{x}_1^*, \ldots, \mathbf{x}_{k-1}^*$ from the distribution $T(\mathbf{y}, \cdot)$, and let $\mathbf{x}_k^* = \mathbf{x}$.
3. Accept $\mathbf{y}$ with probability

$$r_g = \min \left\{ 1, \frac{w(\mathbf{y}_1, \mathbf{x}) + \cdots + w(\mathbf{y}_k, \mathbf{x})}{w(\mathbf{x}_1^*, \mathbf{y}) + \cdots + w(\mathbf{x}_k^*, \mathbf{y})} \right\} \tag{3}$$

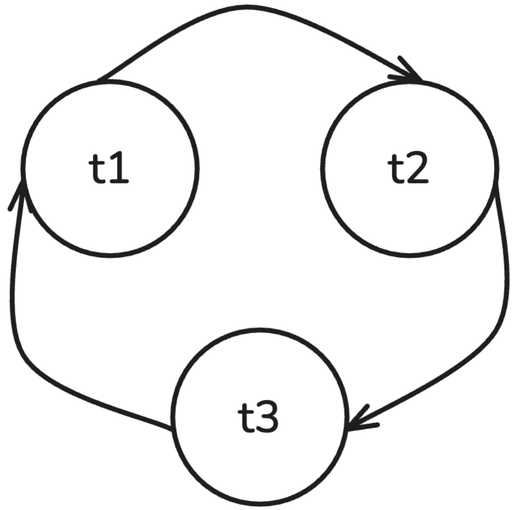
and reject it with probability $1 - r_g$. The quantity $r_g$ is called the *generalized M–H ratio.*

# Results

## Toy Language Model

Consider an linear cycle ***Ising model*** with a binary vocabulary and one-dimensional embeddings

Allows us to brute-force the exact distribution for small sequence lengths



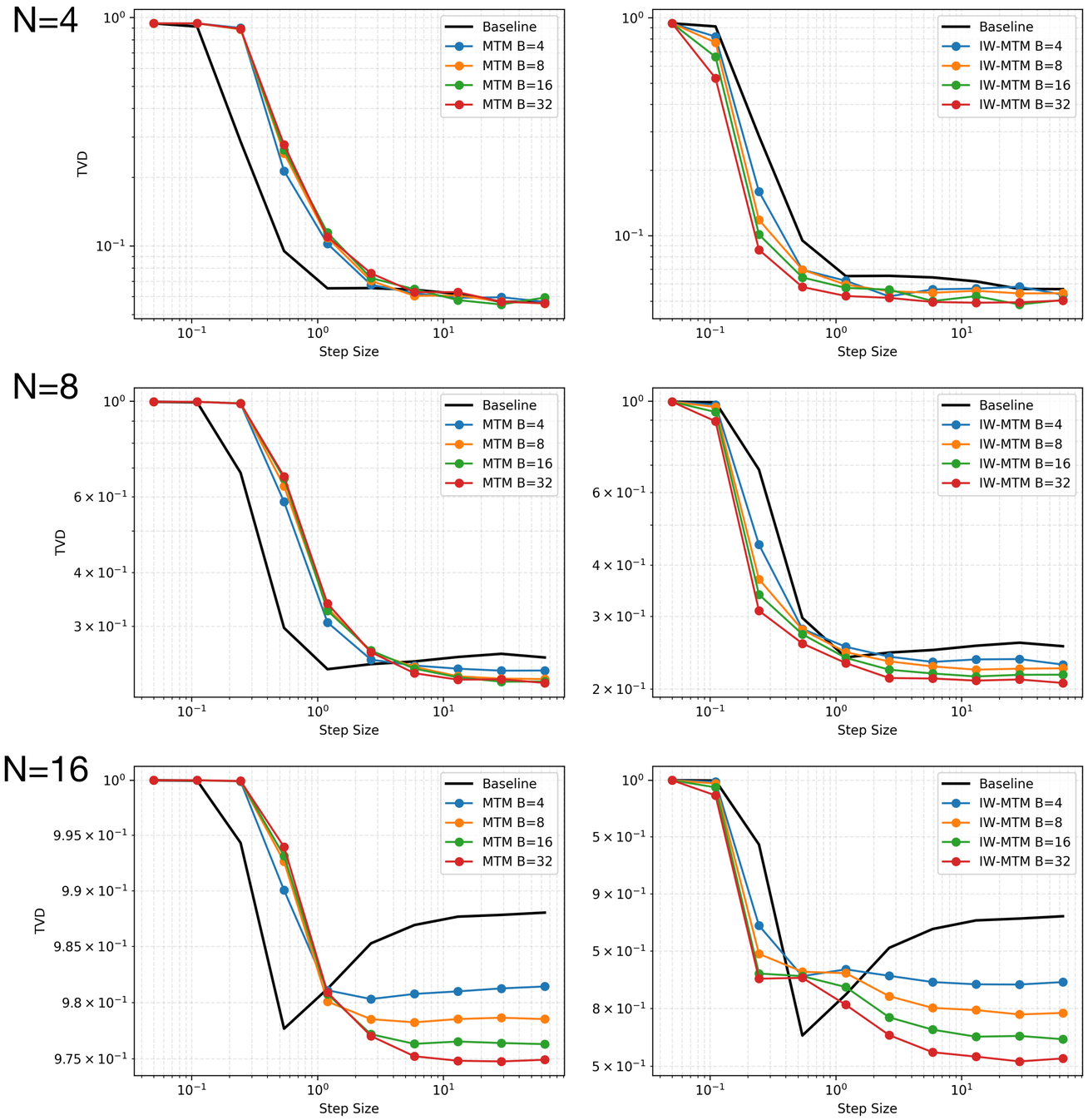
# Results

## Toy Language Model

Scaling w.r.t step size is unique

***Importance-weighted*** variant slightly more efficient

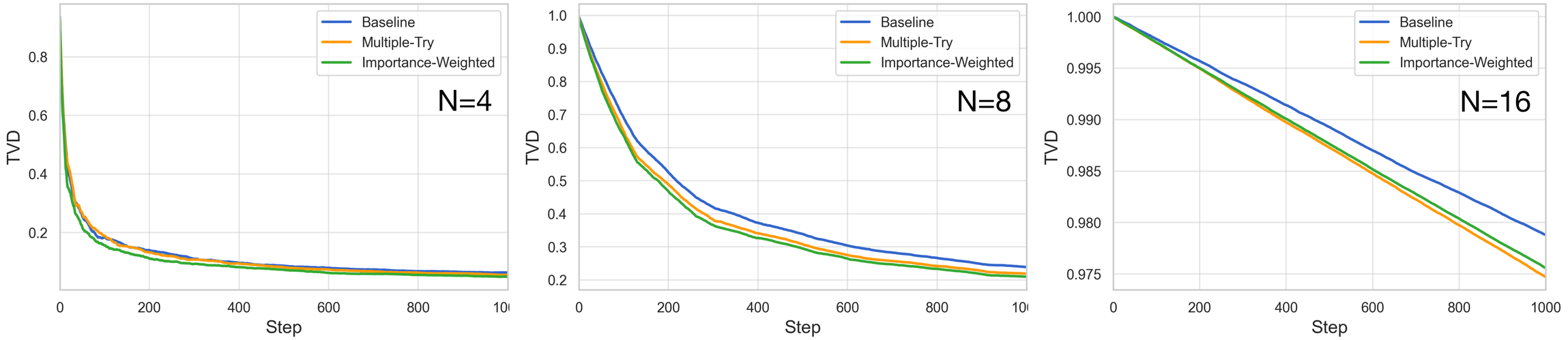
Going from $4 \rightarrow 8 \rightarrow 16$ increases difficulty exponentially



# Results

## Toy Language Model

***Small but consistent wins!***



# Results

## “Large” Language Model

GPT-2 (124M parameters)

Task is ***unconditional sampling*** so the energy function is trivially $-\log p_{\text{LLM}}(x)$

**Language Models are Unsupervised Multitask Learners**

**Alec Radford** [*1] **Jeffrey Wu** [*1] **Rewon Child** [1] **David Luan** [1] **Dario Amodei** [**1] **Ilya Sutskever** [**1]

**Abstract**

Natural language processing tasks, such as question answering, machine translation, reading comprehension, and summarization, are typically approached with supervised learning on task-specific datasets. We demonstrate that language models begin to learn these tasks without any explicit supervision when trained on a new dataset of millions of webpages called WebText. When conditioned on a document plus questions, the answers generated by the language model reach 55 F1 on the CoQA dataset - matching or exceeding the performance of 3 out of 4 baseline systems without using the 127,000+ training examples. The capacity of the language model is essential to the success of zero-shot task transfer and increasing it improves performance in a log-linear fashion across tasks. Our largest model, GPT-2, is a 1.5B parameter Transformer that achieves state of the art results on 7 out of 8 tested language modeling datasets in a zero-shot setting but still underfits WebText. Samples from the model reflect these improvements and contain coherent paragraphs of text. These findings suggest a promising path towards building language processing systems which learn to perform tasks from their naturally occurring demonstrations.

### 1. Introduction

Machine learning systems now excel (in expectation) at tasks they are trained for by using a combination of large datasets, high-capacity models, and supervised learning (Krizhevsky et al., 2012) (Sutskever et al., 2014) (Amodei et al., 2016). Yet these systems are brittle and sensitive to slight changes in the data distribution (Recht et al., 2018) and task specification (Kirkpatrick et al., 2017). Current systems are better characterized as narrow experts rather than competent generalists. We would like to move towards more general systems which can perform many tasks – eventually without the need to manually create and label a training dataset for each one.

The dominant approach to creating ML systems is to collect a dataset of training examples demonstrating correct behavior for a desired task, train a system to imitate these behaviors, and then test its performance on independent and identically distributed (IID) held-out examples. This has served well to make progress on narrow experts. But the often erratic behavior of captioning models (Lake et al., 2017), reading comprehension systems (Jia & Liang, 2017), and image classifiers (Alcorn et al., 2018) on the diversity and variety of possible inputs highlights some of the shortcomings of this approach.

Our suspicion is that the prevalence of single task training on single domain datasets is a major contributor to the lack of generalization observed in current systems. Progress towards robust systems with current architectures is likely to require training and measuring performance on a wide range of domains and tasks. Recently, several benchmarks have been proposed such as GLUE (Wang et al., 2018) and decaNLP (McCann et al., 2018) to begin studying this.

Multitask learning (Caruana, 1997) is a promising framework for improving general performance. However, multitask training in NLP is still nascent. Recent work reports modest performance improvements (Yogatama et al., 2019) and the two most ambitious efforts to date have trained on a total of 10 and 17 `(dataset, objective)` pairs respectively (McCann et al., 2018) (Bowman et al., 2018). From a meta-learning perspective, each `(dataset, objective)` pair is a single training example sampled from the distribution of datasets and objectives. Current ML systems need hundreds to thousands of examples to induce functions which generalize well. This suggests that multitask training many need just as many effective training pairs to realize its promise with current approaches. It will be very difficult to continue to scale the creation of datasets and the design of objectives to the degree that may be required to brute force our way there with current techniques. This motivates exploring additional setups for performing multitask learning.

The current best performing systems on language tasks

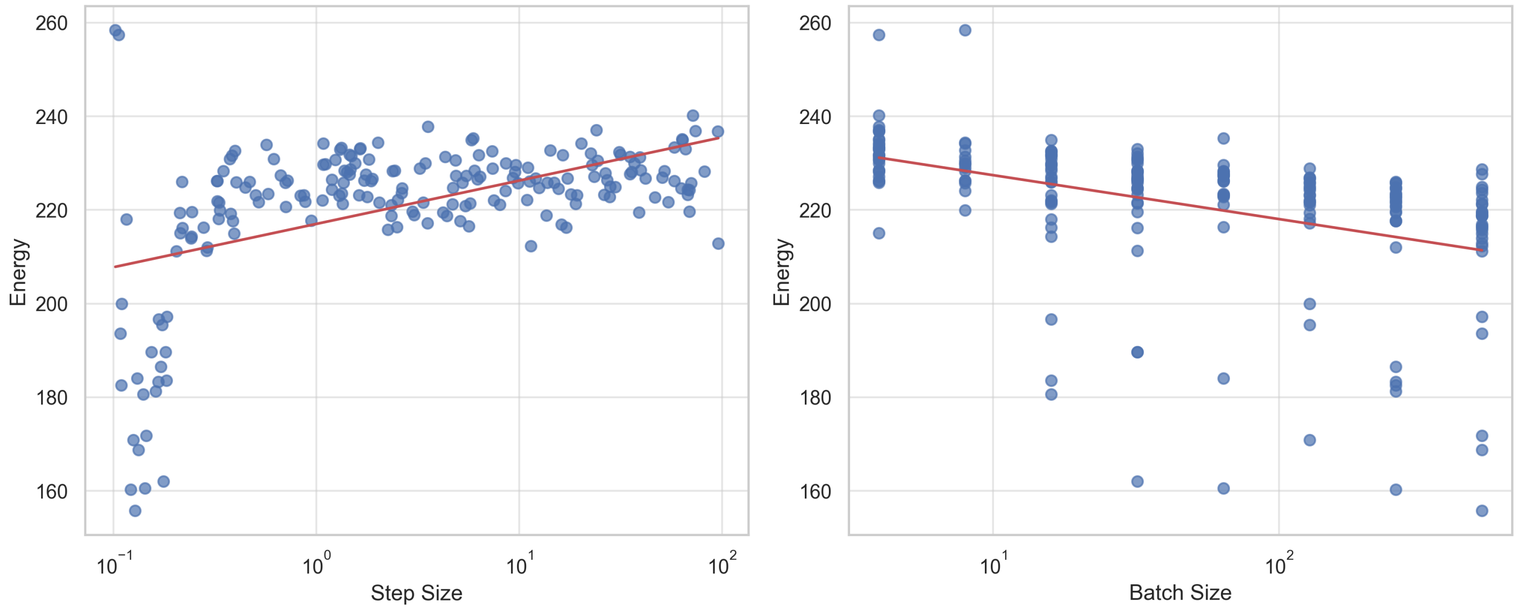
[*,**]Equal contribution [1]OpenAI, San Francisco, California, United States. Correspondence to: Alec Radford <alec@openai.com>.

# Results

## “Large” Language Model

Run ~500 sweeps to find best hyper-parameters

Both methods do best with *large* batches (512) and *small* step size (0.1-0.2)

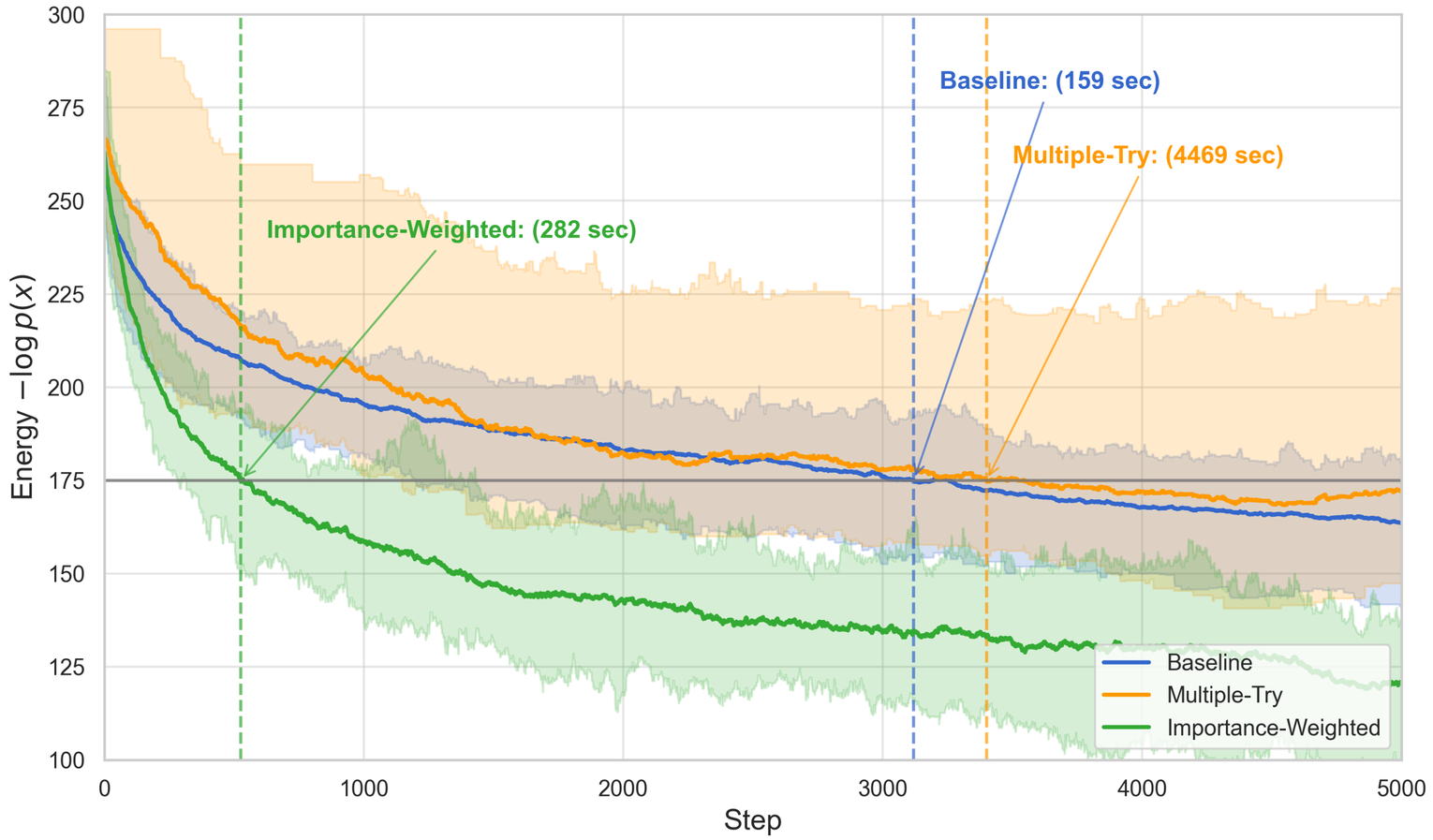


# Results

## “Large” Language Model

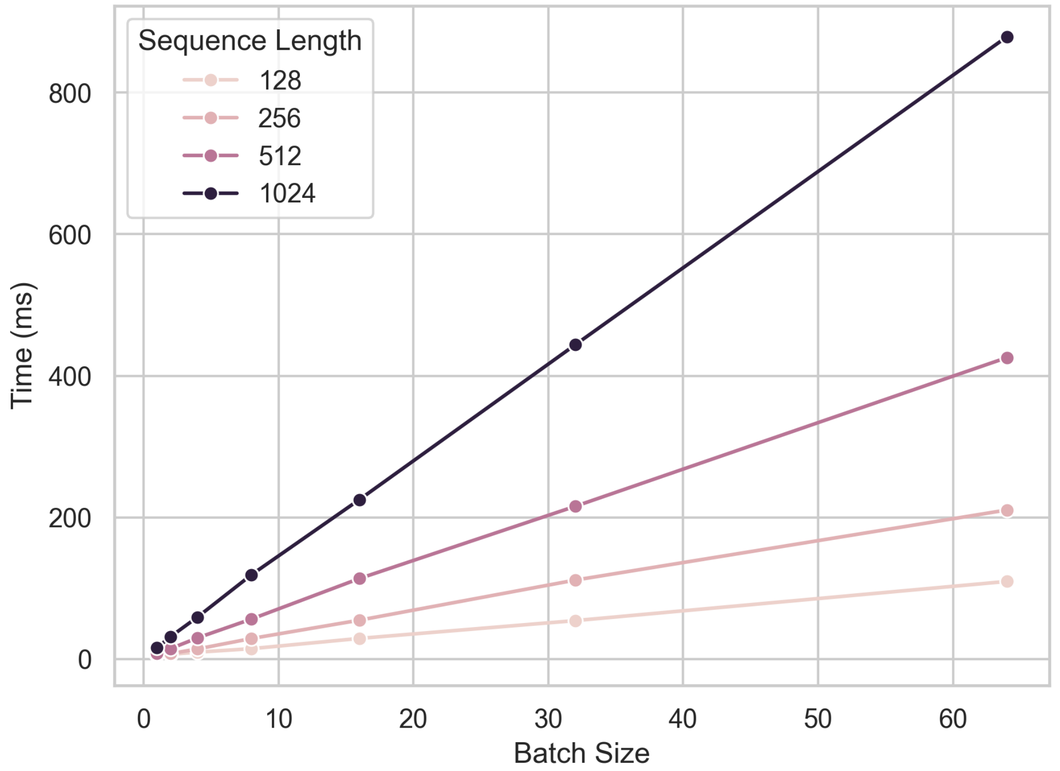
Naive multiple-try is *slow* and *inefficient*

Importance weighting helps with sample efficiency but is about 2x slower per step



# Limitations

Batching isn’t as fast as we might want…



# Limitations

Batching isn’t as fast as we might want…

Batching introduces unsavory ***memory*** constraints when computing the proposal distribution

```python
def pncg_dist_p2(
    embeddings: torch.Tensor,
    state_embeddings: torch.Tensor,
    gradients: torch.Tensor,
    alpha: float = 1.0,
    **_,
):
    # memory-efficient implementation of pncg_dist for p=2
    B, N, E = state_embeddings.shape
    V, _ = embeddings.shape

    grads_flat = gradients.view(-1, E)
    term1 = (grads_flat @ embeddings.T).view(B, N, V)
    term2 = (gradients * state_embeddings).sum(dim=-1).unsqueeze(-1)
    means = -1/2 * (term1 - term2)

    state_emb_flat = state_embeddings.view(-1, E)
    dot_prod = (state_emb_flat @ embeddings.T).view(B, N, V)
    state_emb_norm_sq = (state_embeddings**2).sum(dim=-1).unsqueeze(-1)
    emb_v_norm_sq = (embeddings**2).sum(dim=-1).view(1, 1, V)
    dist_sq = emb_v_norm_sq + state_emb_norm_sq - 2 * dot_prod
    dist_sq = torch.clamp(dist_sq, min=1e-9)
    norms = -1/(2*alpha) * torch.sqrt(dist_sq)

    return F.log_softmax(means + norms, dim=-1) # (B, N, V)
```

# Limitations

Batching isn’t as fast as we might want…

Batching introduces unsavory ***memory*** constraints when computing the proposal distribution

***We can’t always just buy more lottery tickets!***