Enabling Approximate Joint Sampling in Diffusion LMs

Parikshit Bansal *
UT Austin

Sujay Sanghavi
UT Austin

Abstract

In autoregressive language models, each token is sampled by conditioning on all the past tokens; the overall string has thus been sampled from the correct underlying joint distribution represented by the model. In contrast, masked diffusion language models generate text by unmasking tokens out of order and potentially in parallel. Generating an overall string sampled from the correct underlying joint distribution would (again) require exactly one token unmasking in every full-model forward pass. The more tokens unmasked in parallel, the further away the string is from the true joint; this can be seen in the resulting drop in accuracy (but, increase in speed). In this paper we devise a way to *approximately* sample multiple tokens from the joint distribution in a single full-model forward pass; we do so by developing a new lightweight single-layer "sampler" on top of an existing large diffusion LM. One forward pass of the full model can now be followed by multiple forward passes of only this sampler layer, to yield multiple unmasked tokens. Our sampler is trained to mimic exact joint sampling from the (frozen) full model. We show the effectiveness of our approximate joint sampling for both pretrained-only (Dream-7B-Base, Llada-7B-Base) and instruction-tuned (Dream-7B-Instruct, Dream-7B-Coder) models on downstream tasks (GSM8k, MATH, MBPP, HEval) and on unconditional language modeling. When eight tokens are unmasked for each full-model denoising step, our sampling algorithm achieves a MAUVE score of 0.84 (vs marginal baseline of 0.19) with respect to the true joint distribution.

1 Introduction

Masked diffusion language models [16, 2, 13] involve generating text strings by starting from an all-masked sequence of tokens, and then iteratively replacing the masked tokens with tokens from the vocabulary, with each "denoising" forward pass unmasking one or a few tokens. As opposed to auto-regressive models which generate tokens left to right and one token in each forward pass, in masked diffusion models tokens can be potentially unmasked in any order and also potentially multiple tokens can be unmasked in parallel.

The higher the number of tokens unmasked in parallel after a single denoising forward pass, the faster and cheaper the overall generation [16]. However, the quality of the resulting generation is also generally seen to be worse. State-of-the-art large masked diffusion models (7B parameters) are advertised [21, 14, 18, 8] to achieve competitive downstream task performance, when compared to their auto-regressive counterparts; however, this parity is only achieved via slow one-token-per-denoise-step generation from the masked diffusion model. On decreasing the number of denoising steps, there is a sharp decrease in their performance. For example, Dream-7B-Instruct model's accuracy drops from 85% accuracy to 77% when generating two tokens instead of one per denoising step [6].

*parikshitb52@gmail.com

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

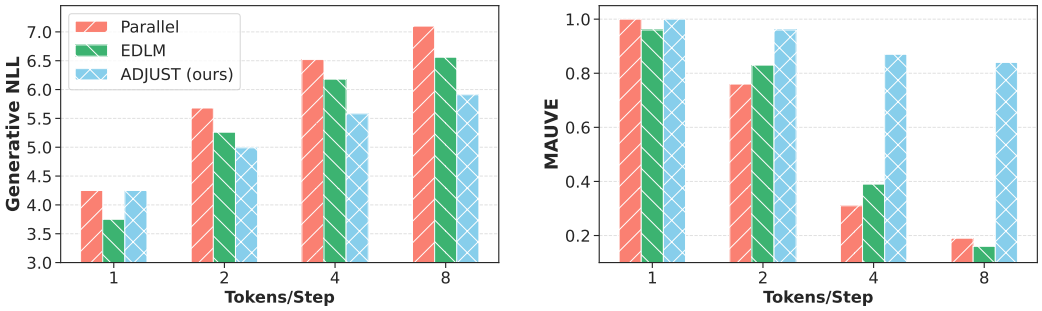

Figure 1: In this figure, we use Dream-7B-Base to generate a token string of length 128 tokens starting from an all masked string (kindly refer to Sec 4.2 for details). We report the generative negative log-likelihood (lower is better) and the MAUVE score (higher is better) for the generated strings. We vary the number of tokens sampled per diffusion denoising step from one to four. We observe an increase in gen. NLL and a decrease in the MAUVE score as more tokens are generated in parallel. Our method ADJUST achieves the best NLL and MAUVE (as compared to baseline methods like naive parallel sampling, and energy-based models).

Our goal is to alleviate the drop in performance seen during parallel generation, while still unmasking multiple tokens in each denoising forward pass of the full masked diffusion language model. Our **main intuition** is that a big reason vanilla parallel generation is inaccurate because it is sampling multiple tokens from the product of their conditional marginal distributions, instead of sampling them from the true joint distribution the model encodes. In particular, the output token probabilities from the forward pass of diffusion model represent per-token marginals, and vanilla parallel generation involves sampling multiple of these independent of each other (we formalize this intuition later). This stands in contrast to auto-regressive models, where every token is correctly conditioned on all the ones generated before it – left-to-right generation in auto-regressive models samples from the true distribution.

Similar exact sampling in masked diffusion language models would require exactly one token unmasked in every forward pass. Our **main contribution** is devising a new way to do **approximate sampling** of multiple tokens from their joint distribution – by unmasking them one at a time using a small auxiliary "approximate sampler" model that undertakes many forward passes after every single denoising forward pass of the main big masked diffusion language model. This allows the sampling of our tokens to be at least informed by the specific identities of the other tokens unmasked before them in the same big-model forward pass.

We term our method as ADJUST (Sec 3.2). ADJUST is a **baby version of the base diffusion model**, that is added on to the frozen base model. Our model reuses the hidden states of the base model, and is trained to so that its forward passes mimic those of the base model. Importantly, ADJUST is not a separate distilled diffusion model; rather, it is designed to work together with the base model. Algorithmically, each diffusion denoising step with access to ADJUST corresponds to a single forward pass of the base diffusion, followed by $K-1$ forward passes of the ADJUST module. To work in tandem with the base diffusion model, ADJUST's architecture and training recipe have been carefully designed. The objective of all of these is for the output of ADJUST to mimic true (slow) joint sampling as closely as possible.

Contributions:

1. Each denoising forward pass of a masked diffusion model provides every token a marginal distribution for that token given the current input; there is no way to jointly sample even two tokens with one forward pass (Sec 3.1). True joint sampling is possible only if exactly one token is unmasked in every denoising step. Evidence of the value of joint sampling can be seen in existing work, where it shows up as increased accuracy with increased denoising steps. We develop a new architecture that enables approximate joint sampling of a few tokens at a time for an existing off-the-shelf masked diffusion model. Section 3.2 develops the probabilistic view motivating our method.

2. Our approximate joint sampler consists of a single layer "on top of" the diffusion model; we call this ADJUST. Approximate joint sampling is achieved by executing multiple forward passes of only this small ADJUST, for a single forward pass of the big base model. Each forward pass of the ADJUST augments the input to the next forward pass of the ADJUST. Section 3.2 describes in detail our architecture.

3. Training the single-layer sampler model involves a series of non-trivial choices guided by our probabilistic view. This is because each forward pass of the small sampler needs to mimic an actual forward pass of the full model. This needs realistic inputs to the drafter at various noise levels, which in turn have to be offline generated by the base model. Section 3.4 provides the details of how this is achieved.

4. We show the utility of our method ADJUST on both *unconditional generation* (Sec 4.2) and downstream evaluation (Sec 4.3). Across models, sampling parameters and evaluation settings, ADJUST yields consistent gains compared to naive parallel sampling. For example, when considering four tokens generated per time step, for the Dream-Base model, ADJUST increases GSM8k accuracy by 20 percentage points and MAUVE score by 0.5 points. In the absence of domain-specific prompts, a generically trained (assuming input prompt to be null) ADJUST, is able to generalize and perform well on downstream tasks as well.

2 Related Work

Discrete diffusion language models (LMs) [16, 17, 13] are a promising alternative to auto-regressive (AR) LMs. Large diffusion LMs [21, 14] competing with the AR counterparts are based on an absorbing-state Markov process, which noises tokens into a "masked" state. [23] show that masked diffusion language models can also be considered as mask denoisers. [15] show that latent conditioning on important "anchor" tokens leads to much better performance.

In order to further the capabilities of diffusion LMs [11, 6] propose to use both AR and diffusion models in tandem to get final logits. [6, 4] propose training-free decoding methods for any-order models with the goal of improving token latency. Other works [19] introduced KV cache for masked diffusion LMs and adaptively decide the number of tokens to decode per denoising step. [3, 1] train specialized models which are cheaper to infer. Our work instead proposes parallel decoding as joint sampling from the underlying distribution of diffusion model and proposed to train a baby model to work in tandem for the task.

3 Method

Notation We denote our input $\boldsymbol{x} = [x^0, x^1, \ldots, x^{L-1}] \in \mathcal{V}^L$ as a string of L tokens in vocabulary $\mathcal{V}$. We let $\mathrm{M} \in \mathcal{V}$ denote a special mask token in the vocabulary. Positions having M token are considered "missing" and have to be appropriately filled in with non-mask tokens in $\mathcal{V}/\mathrm{M}$. We let $\mathcal{M}(\boldsymbol{x})$ denote the indices (in string $\boldsymbol{x}$) of these masked positions. Let $\boldsymbol{x} \oplus x^i$ denote a new string where the ith position of $\boldsymbol{x}$ (implicitly assumed to be a masked position) is filled with the token value x^i. Let masked diffusion models be represented by the pair $(f, \boldsymbol{W})$. Here $f : \mathcal{V}^L \to \mathbb{R}^{d \times L}$ [2] maps from an input string of L tokens $\boldsymbol{x}$ to an output of L embeddings, with $f^i(\boldsymbol{x})$ denoting the output embedding at the ith position. $\boldsymbol{W} \in \mathbb{R}^{\mathcal{V} \times d}$ denotes the language model (LM) head that maps an embedding to logits. Additionally, let $p^i(\cdot \mid \boldsymbol{x})$ denote the marginal probabilities at position i conditioned on $\boldsymbol{x}$ and p_* denote the true underlying distribution of the diffusion model (we develop these concepts in Sec 3).

Background. We study masked diffusion models with "unmask and commit" inference; that is, the tokens to be generated are initially started out as M, and these locations are iteratively filled in with other tokens from the vocabulary via forward passes of the model. Unconditional generation would thus involve a series of strings $\phi \to \boldsymbol{x}_1 \to \boldsymbol{x}_2 \to \ldots \to \boldsymbol{x}_T$, where $\phi = \boldsymbol{x}_0$ is the all-mask sequence and $\boldsymbol{x}_T$ is the completely unmasked string.

[2] we assume that the time information is implicit in the masked input sequence [23]

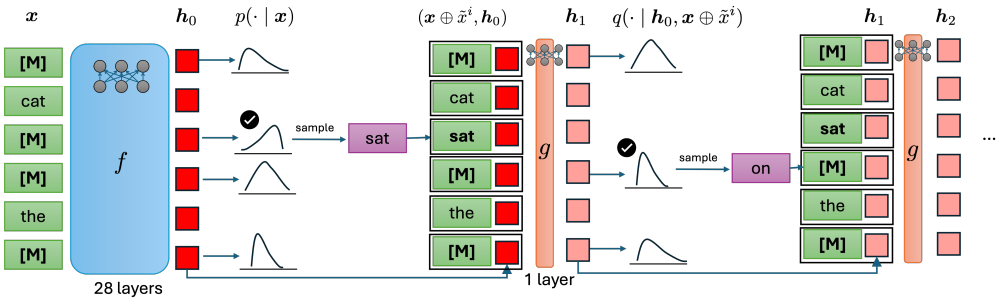

Figure 2: **ADJUST.** The figure shows one forward pass of the base diffusion models followed by two passes of the ADJUST sampling module. The tick in the figure denotes the position selection logic (TOP in Algo 2). ADJUST module g is a single layer bi-direction attention based diffusion LM. After unmasking a single token (`sat` in the figure), the module g conditions the prior output embeddings $\boldsymbol{h}_0$ on the sampled token before proceeding to sample the next token `on`. Notation in figure follows Algo 2.

3.1 Problem Setup

Let $(f, \boldsymbol{W})$ denote a frozen base masked diffusion model. We will make a joint sampler for $(f, \boldsymbol{W})$. Recall that discrete diffusion models only output per-token marginals [16, 2]; that is for each position $i = 1, \ldots, L$ a single forward pass yields a probability distribution $p^i(\cdot|\boldsymbol{x}) = \text{softmax}[\boldsymbol{W} f^i(\boldsymbol{x})]$ over the vocabulary $\mathcal{V}$. If multiple tokens are unmasked in the same forward pass, this is done by independently generating each of them from their respective marginals. Furthermore, let p_* denote the model's true underlying joint distribution. Intuitively, for a perfectly trained model this is data distribution used to train $(f, \boldsymbol{W})$.

True underlying joint distribution. If we consider the M tokens as nulls, then for a fixed output position i we can interpret the model's output $p^i(x^i|\boldsymbol{x})$ as the conditional distribution of that token i, given the unmasked tokens in $\boldsymbol{x}$; thus, for any single position i, $p_*(x^i|\boldsymbol{x}) = p^i(x^i|\boldsymbol{x})$. Then, exact unconditional generation of a length-L fully unmasked string $\boldsymbol{x}$ from this p_* can be written as the following via chain rule (with σ as some permutation of $[L]$):

$$p_*(\boldsymbol{x}) = \prod_{k=1}^{L} p_*(x^{\sigma(k)} \mid \phi \oplus \ldots \oplus x^{\sigma(k-1)}) = \prod_{k=1}^{L} p^{\sigma(k)}(x^{\sigma(k)} \mid \phi \oplus \ldots \oplus x^{\sigma(k-1)}) \tag{1}$$

However, this corresponds to sampling *exactly* one token at a time – with the unmasking order given by σ of the L positions.

The product marginal problem. While one-token-per-forward-pass unmasking represents exact sampling, it is also very slow and expensive; existing work advocates sampling multiple tokens in parallel with one forward pass. However, this corresponds to sampling from the product of their individual marginals; for example, if K tokens $x^{\sigma(1)}, \ldots, x^{\sigma(K)}$ are generated in parallel in the first step after ϕ, their joint distribution satisfies

$$p_{parallel}\left(x^{\sigma(1)}, \ldots, x^{\sigma(K)}\right) \quad = \quad \prod_{k=1}^{K} p^{\sigma(k)}\left(x^{\sigma(k)} \mid \phi\right) \tag{2}$$

Thus, parallel generation results in a sequence drawn from a different distribution from the true p_*; our paper is based on the idea that this is the cause of the (widely recognized) loss in accuracy when multiple tokens are sampled in parallel.

3.2 Our method : ADJUST

We develop a new method to generate K tokens with one forward pass of the base diffusion model $(f, \boldsymbol{W})$. We do this by sampling one token at a time from a small extra "baby diffusion" draft model (see Fig 2). Crucially, mimicking one token at a time sampling, the token unmasked after each step of ADJUST is fed back in to the input for the next step of ADJUST. This allows for each unmasking

Algorithm 1 ADJUST Training (Sec 3.4)

Input: $\boldsymbol{x} \sim \mathcal{D}$ input strings, (frozen) diffusion model $f, \boldsymbol{W}$ (where $\boldsymbol{W}$ is the LM head)
repeat
SAMPLE TRAINING STRINGS (SEC 3.4)
  $\boldsymbol{x} \leftarrow$ final string after a random number of denoising steps by base model on $\boldsymbol{x}$
  $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_K \leftarrow$ strings after further K denoising steps by base model on $\boldsymbol{x}$
  $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_K \leftarrow f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_K)$
  $\hat{\boldsymbol{h}}_0 \leftarrow f(\boldsymbol{x})$
UNROLL EMBEDDINGS (EQN 3)
  for $k = 0$ to $K - 1$ **do**
    $\hat{\boldsymbol{h}}_{k+1} \leftarrow g_\theta(\hat{\boldsymbol{h}}_k, \boldsymbol{x}_{k+1})$
COMPUTE LOSS (EQN 5)
  $\mathcal{L}(\theta) \leftarrow \sum_{k=1}^{K} D_{\mathrm{KL}}(\boldsymbol{W}\boldsymbol{h}_k \,||\, \boldsymbol{W}\hat{\boldsymbol{h}}_k)$
  Take gradient step on $\nabla_\theta \mathcal{L}(\theta)$
until converged
Return: g_θ

Algorithm 2 ADJUST: Sampling (Sec 3.3)

Input: Prompt $\boldsymbol{x}$, frozen discrete diffusion model $f, \boldsymbol{W}$ (where $\boldsymbol{W}$ is the LM head), ADJUST g, number K of tokens to generate in each step, base model's unmasking logic TOP
while $|\mathcal{M}(\boldsymbol{x})| > 0$ **do**
  $\boldsymbol{h}_0 \leftarrow f(\boldsymbol{x})$
  for $k = 0$ to $K - 1$ **do**
SELECT POSITION TO UNMASK
    $i \leftarrow \mathrm{TOP}\{\boldsymbol{W}\boldsymbol{h}_k^j : j \in \mathcal{M}(\boldsymbol{x})\}$
SAMPLE TOKEN, UPDATE TOKEN STRING
    $\tilde{x}^i \sim \mathrm{softmax}(\boldsymbol{W}\boldsymbol{h}_k^i)$
GET UPDATE TOKEN STRING
    $\boldsymbol{x} \leftarrow \boldsymbol{x} \oplus \tilde{x}^i$
UPDATE $\boldsymbol{h}$ ON TOKEN $\tilde{x}^i$ (EQN 3)
    $\boldsymbol{h}_{k+1} \leftarrow g(\boldsymbol{h}_k, \boldsymbol{x})$
Return: $\boldsymbol{x}$

to be informed by the identities of the other tokens unmasked before it; in contrast, simple parallel sampling results in each token being sampled independent of the identities of the other tokens. We illustrate this in Fig 2. Importantly, ADJUST conditions on the tuple of updated token string and most recent output embeddings (elaborated on later). This enables ADJUST to work in tandem with the base model and differentiates it from a stand-alone distilled diffusion LM.

ADJUST The ADJUST model is represented as the pair $(g, \boldsymbol{W})$. Different from f, the function g operates on a pair of inputs $g : \mathbb{R}^{d \times L} \times \mathcal{V}^L \to \mathbb{R}^{d \times L}$. Let $\boldsymbol{h} \in \mathbb{R}^{d \times L}$ denote the set of "most current" embeddings (Sec 3.3) and let $\boldsymbol{x}$ be the current unmasked string. Then for each position i, ADJUST yields a probability distribution :

$$q^i(\cdot \mid \boldsymbol{h}, \boldsymbol{x}) = \mathrm{softmax}\big[\boldsymbol{W} g^i(\boldsymbol{h}, \boldsymbol{x})\big]$$

We draw the reader's attention to the parallels between q and p, where the only difference between the two is the inclusion of the extra embeddings $\boldsymbol{h}$ as input for g. This allows g to be much smaller model than f.

Architecturally, ADJUST first embeds the input token string $\tilde{x}^{\sigma(1)} \oplus \ldots \oplus \tilde{x}^{\sigma(k-1)}$ using the frozen base model's token embeddings. It then concatenates these token embeddings with the set of embeddings $\boldsymbol{h}$ given as input. The concatenated embeddings are then down-projected into the base model's hidden dimension size and then passed through a transformer decoder layer. For simplicity, ADJUST model's decoder layer has the same architecture as the base model's decoder layers. We refer the reader to Fig 5 (in appendix) for an architecture diagram of g.

3.3 Inference with ADJUST

Let $\boldsymbol{x}$ be the current unmasked string and suppose we are sampling K tokens per denoising step. Let $\sigma(1), \ldots, \sigma(K)$ be the set of positions of the tokens that will be sampled. We now describe how inference proceeds with ADJUST.

Current Embeddings are the output embeddings of the most recent forward pass of either the base model f or ADJUST module g. Concretely, let $\boldsymbol{h}_0 = f(\boldsymbol{x})$, define the most recent embedding after doing a forward pass with the base diffusion model. Let $\boldsymbol{h}_k$ after k forward passes of the ADJUST module. Then they can be recursively defined as :

$$\boldsymbol{h}_{k+1} = g\left(\boldsymbol{h}_k \,,\, \boldsymbol{x} \oplus \tilde{x}^{\sigma(1)} \oplus \ldots \oplus \tilde{x}^{\sigma(k+1)}\right) \tag{3}$$

where $\tilde{x}^{\sigma(k)}$ represents sampled token for position $\sigma(k)$. The algorithm for doing one denoising step is : (also refer Algo 2)

(1) Do a forward pass of the base diffusion model and set $\boldsymbol{h}_0 = f(\boldsymbol{x})$.

(2) Sample the first token $\tilde{x}^{\sigma(1)}$ from base model's marginals $p^{\sigma(1)}(\cdot \mid \boldsymbol{x}) = \mathrm{softmax}[\boldsymbol{W}\boldsymbol{h}_0^{\sigma(1)}]$

(3) Set the current unmasked string to $\boldsymbol{x} \oplus \tilde{x}^{\sigma(1)}$ and current embedding to $\boldsymbol{h}_0$.

(4) Do forward pass of the ADJUST model $\boldsymbol{h}_1 \leftarrow g\left(\boldsymbol{h}_0\,,\, \boldsymbol{x} \oplus \tilde{x}^{\sigma(1)}\right)$.

(5) Sample the second token $\tilde{x}^{\sigma(2)}$ from new marginals $\mathrm{softmax}[\boldsymbol{W}\boldsymbol{h}_1^{\sigma(2)}]$. Repeat step 3-5 with updated counter.

Note that the only difference between naive parallel and ADJUST sampling is updating the output embeddings (line 8). The final probability for sampling K tokens in a single denoising step is:

$$p_{\text{ADJUST}}(\tilde{x}^{\sigma(1)}, \ldots, \tilde{x}^{\sigma(K)} \mid \boldsymbol{x}) \;=\; \underbrace{p^{\sigma(1)}\left(\tilde{x}^{\sigma(1)} \mid \boldsymbol{x}\right)}_{\text{Sample token from base model probs}} \times \prod_{k=2}^{K} \underbrace{q^{\sigma(k)}\left(\tilde{x}^{\sigma(k)} \mid \boldsymbol{h}_{k-2}\,,\, \boldsymbol{x} \oplus \tilde{x}^{\sigma(1)} \oplus \ldots \oplus \tilde{x}^{\sigma(k-1)}\right)}_{\text{Sample tokens from ADJUST probs by iterating K}-1\text{ times.}} \tag{4}$$

3.4 Training Methodology

Training Loss. We want our joint approximation p_{ADJUST} (Eqn 4) to mimic the true joint distribution p_* (Eqn 1). Notice that if $q\left(\cdot \mid \boldsymbol{h}_{k-2}\,,\, \boldsymbol{x} \oplus \tilde{x}^{\sigma(1)} \oplus \ldots \oplus \tilde{x}^{\sigma(k-1)}\right)$ was equal to $p\left(\cdot \mid \boldsymbol{x} \oplus \tilde{x}^{\sigma(1)} \oplus \ldots \oplus \tilde{x}^{\sigma(k-1)}\right)$ for all values of $k \geq 2$, p_{ADJUST} (Eqn 4) would be exactly equal to p_* (Eqn 1). Hence our training objective minimizes the KL divergence between p and q. Our training loss is hence a sum of KL divergence terms between .

$$\mathcal{L}(\theta) \;=\; \sum_{k=2}^{K} D_{\text{KL}}\left[p\left(\cdot \mid \boldsymbol{x} \oplus \ldots \oplus \tilde{x}^{\sigma(k-1)}\right) \,||\, q\left(\cdot \mid \boldsymbol{h}_{k-2}, \boldsymbol{x} \oplus \ldots \oplus \tilde{x}^{\sigma(k-1)}\right)\right] \tag{5}$$

Note that probabilities $p(\cdot \mid \boldsymbol{x})$ and $q(\cdot \mid \boldsymbol{h}, \boldsymbol{x})$ represent probabilities for each position in the sequence as p^l and q^l. The KL divergence term is computed for each token position and not just for a single token position. The loss is represented in line 9 of Algo 1.

Generating Training Samples. We describe how we sample partially masked token strings utilized in Eqn 5 for training.

1) We first realize that q would be evaluated at test on the distribution of partially masked strings encountered during denoising with the base diffusion model p. We consider $\boldsymbol{x}$ to be exactly the partially masked strings formed by the base model's denoising process. That is, given an initial distribution of masked strings, we first sample an initial *prompt* string. We then run the base model's denoising process to completion. Note that here we are using the base model's unmasking logic (see App E for details) and perform single token $K = 1$ unmasking to sample from the true joint. The history of all partially masked strings encountered during the denoising process serve as our distribution of $\boldsymbol{x}$.

2) Following the same logic, we argue that σ for a training sample $\boldsymbol{x}$ should be the next K positions generated by the base model starting from $\boldsymbol{x}$ (on single token unmasking). Finally we describe how to get $\boldsymbol{h}_k$. Recall that, $\boldsymbol{h}_k$ is defined as $g\left(\boldsymbol{h}_{k-1}\,,\, \boldsymbol{x} \oplus \tilde{x}^{\sigma(1)} \oplus \ldots \oplus \tilde{x}^{\sigma(k)}\right)$. We use the recursive computation of embeddings with teacher forcing of token inputs during training (line 8 of Algo 1). The "unrolling" of g during training is considered helpful by prior work [10, 9].

Putting these things together, we present our training algorithm in Alg 1.

4 Experiments

4.1 Experimental Setup

We consider four different state-of-the-art discrete diffusion language models as base models for our empirical investigation. We consider a pretrained-only model (Dream-7B-Base, Llada-7B-Base) and an instruction-tuned model (Dream-7B-Instruct, Dream-7B-Coder) [21]. Recall that generating

		K=1		K=2		K=4		K=8	
		NLL	Mauve	NLL	Mauve	NLL	Mauve	NLL	Mauve
T=0.6	Parallel	1.11	1.00	1.61	0.89	2.21	0.29	2.21	0.02
	EDLM	**0.83**	0.98	1.33	0.94	1.89	0.31	2.05	0.03
	ADJUST **(ours)**	1.11	1.00	**1.23**	**0.97**	**1.64**	**0.79**	**1.66**	**0.67**
T=1.0	Parallel	4.25	1.00	5.68	0.76	6.52	0.31	7.10	0.19
	EDLM	**3.75**	0.96	5.26	0.83	6.18	0.39	6.56	0.16
	ADJUST **(ours)**	4.25	1.00	**4.99**	**0.96**	**5.58**	**0.87**	**5.91**	**0.84**

Table 1: **Unconditional Generation.** In this table we compare the NLL and the MAUVE scores for three different methods : Marginal, Joint and Energy-based (EDLM). The base diffusion model is Dream-Base-7B. Across different number of tokens K sampled per step, and different sampling temperature, joint sampling consistently outperforms both of the baselines, parallel sampling and EDLM.

multiple tokens per denoising step leads to a deviation from the true underlying distribution. We therefore compare performances of different methods as a function of number of tokens K generated for each denoising step. For simplicity of the setup, we generate the same number of tokens for each denoising step. We use the base model's unmasking logic for our experiment (unless specified). Dream models' unmasking logic chooses positions with the least entropy and Llada uses the most confident token.

Methods We briefly describe the methods considered. **1)** *True joint sampling*: This method corresponds to generating a single token $K = 1$ in each denoising step. Recall that we established that generating one token per denoising step corresponds to sampling from the true joint distribution. Joint sampling serves as the oracle performance our method ADJUST aims to mimic. **2)** *Parallel sampling*: This is the naive parallel sampling, default in diffusion models. Each of the K generated tokens is sampled independently from the base model's logits. **3)** *Energy-based model sampling (EDLM)* [20]: operates by drawing multiple token string unmasked completions (two in our experiments). It then chooses the completion with the lowest auto-regressive generative perplexity. The chosen string is then re-masked such that the final string has K less masked positions (see Sec A.1 for discussion of hyper-parameter choices). We use the base model's unmasking logic. **4)** *Dynamic Parallel Decoding (DPD)* [19] : operates by unmasking all positions whose most likely token is above a confidence threshold. This method is suited for decoding in greedy (sampling temperature is 0) parallel settings. Comparison with DPD is presented in Sec 4.4 **5)** ***ADJUST (ours)***: This is our joint approximation algorithm (Sec 3) which updates the logits conditioned on the sampled tokens. We acknowledge other baseline methods like *adaptive parallel decoding*(APD) [6] and *discrete copula diffusion* (DCD) [11] which we omit from the main text. Their formulation is orthogonal to our goal of mimic generation from the true joint distribution. (see Sec B for results and discussion).

Training Details We train ADJUST with a roll-out of three, that is, we assume $K = 4$ for training. All models are trained for 2 epochs with constant learning rate of 5e-5 and batch size of 32. ADJUST has 200M parameters (compared to 7B parameters for base model). Recall that ADJUST assumes an initial distribution of prompts $\mathcal{D}$ (kindly refer Alg 1). For pretrained-only models we assume the initial prompt as just the beginning-of-sentence token, with max generation length of 128 tokens. We run the base model inference with a temperature of 1.0 for 100K samples to get our training dataset. For Dream-Instruct, we subsample 20K unique questions from MetamathQA [22] and generate with max generation length of 256 tokens, with temperature of 0.1 and nucleus of 0.9. For Dream-Coder we similarly subsample 20K prompts from Opencoder-LLM's opc-sft-stage2 [5]. All data-generation runs take one GPU day on H100. All training runs take four GPU days on H100. Note that we are only sampling the prompts and not the ground truth responses.

4.2 Unconditional Generation

We consider generation with the input prompt as the beginning-of-sentence token, with a max generation length of 128. This *unconditional generation* task aims to measure the language modeling capability of the model. We generate the output strings using our baseline methods and compare the

Model	Task	K=1	K=2		K=4		K=8	
			Parallel	ADJ.	Parallel	ADJ.	Parallel	ADJ.
Llada-Base	GSM	68.94	60.61	66.67	54.55	57.58	19.70	28.79
	MBPP	40.40	28.40	37.40	19.40	29.40	6.60	20.00
	HEval	33.54	25.00	29.88	12.80	23.78	5.49	10.37
Dream-Base	GSM	78.40	71.20	75.60	43.20	64.00	11.60	36.80
	MBPP	48.00	40.80	44.80	33.60	33.60	18.40	19.60
Dream-Inst	GSM	85.20	76.80	80.08	58.00	74.00	20.00	44.40
	MATH	32.57	32.29	31.42	15.14	22.28	1.14	5.70
Dream-Coder	MBPP	63.00	47.00	57.40	12.60	16.00	4.60	9.20
	HEval	75.00	55.49	73.17	20.73	57.32	1.83	28.05

Table 2: **Downstream Evaluation.** In this table we report numbers on downstream tasks, GSM8k & MBPP for the Dream-Base (denoted as Base). Since ADJUST for Dream-Instruct (denoted as Inst.) in trained on MetamathQA prompts, we report GSM8k & MATH numbers. Sampling a single token for each denoising step gives the best performance. ADJUST achieves better downstream accuracy tasks across values of tokens sampled in each denoising step K.

negative log-likelihood (NLL) of the generated string as measured by an auto-regressive (AR) model (Qwen-2.5-7B for our setup). Recall that our target is to mimic the base model's true joint distribution. Hence we also measure the generated strings MAUVE score [12] (measured by GPT2-Large) with strings sampled from the true joint distribution, for given set of generation parameters. MAUVE score intuitively captures a notion of KL divergence between a two given sets of sentences and varies between zero and one with higher values implying a lower KL divergence between the distribution of the two sets.

Results are presented in Table 1. We present results for pre-trained only Dream-Base-7B. We evaluate generations for two sampling temperatures, 0.6 and 1.0, highlighting the robustness of our method to sampling parameters. The MAUVE score is computed w.r.t. the samples from diffusion model's true joint distribution, that is, samples generated with one token per diffusion step $K = 1$ sampling. For $K = 1$, EDLM has lower NLL compared to our method, while (seemingly counterintuitively) having lower MAUVE. This is an expected behavior, as EDLM samples two different completions for each denoising step and chooses the completion with a lower NLL. The selection of lower NLL completion biases the EDLM model away from the true joint distribution of the diffusion model and more towards the underlying distribution of the auto-regressive model (used for computing NLL). This is also why EDLM achieves lower NLL than parallel and ADJUST for $K = 1$. Results for $K = 8$ show how well ADJUST generalizes beyond the training rollouts.

4.3 Downstream Evaluation

Here we show that our ADJUST method leads to better downstream accuracy than naive parallel sampling and closes matches performance of $K = 1$ for any degree of parallel generation. We use low temperature sampling (0.0-0.1) for generation. Due to computational constraints we limit our evaluation to 250 question from GSM8K & MBPP and 350 question (50 question per task) for Minerva math. Exact sampling and evaluation parameters are summarized in Sec D.1.

Results Recall that the ADJUST for the pretrained-only model is trained on domain-agnostic data (generations starting from begin-of-sentence token). Hence the results for Dream-Base and Llada-Base show the utility of a generically trained ADJUST for downstream tasks like GSM8K and MBPP in Table 2. Our method is better than marginal across all values of number of generated tokens in parallel even without ever specifically being prompted for math or code data. Since the downstream evaluation utilizes low temperature sampling, EDLM leads to same performance as naive sampling. Hence we don't report those numbers in the table. ADJUST for Dream-Instruct model is trained on MetamathQA prompts. Hence we evaluate on math domain downstream tasks (GSM8K and Minerva-math). Similarly, since we train Dream-Coder on opc-sft-stage2 prompts, we evaluate it on coding tasks. Results for $K = 8$ lie outside the training rollout range of $K = 4$ and demonstrate the generalization of ADJUST along longer prediction horizons.

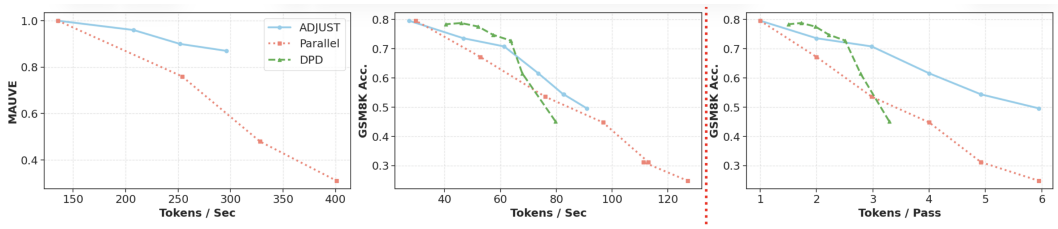

Figure 3: **Throughput evaluation.** In this figure we report tokens produced per second for unconditional (leftmost) and downstream evaluation (middle) on GSM8k for Dream-Base-7B (Sec 4.4). Recall that DPD is a greedy (temperature=0) sampling based method and hence is missing from unconditional generation figure which requires sampling to generate text from base model's underlying distribution. For both evaluation settings, ADJUST demonstrates gain in performance for a given throughput compared to naive parallel sampling. Additionally, while DPD is limited to a smaller range of tokens produced per second, ADJUST achieves higher throughput (>80 toks/sec) while outperforming both parallel sampling and DPD in that regime. The rightmost figure shows comparison with numbers with tokens per pass on the x-axis

4.4 Throughput evaluation

We report numbers on the accuracy of various methods as a function of throughout for Dream-Base-7B model. Evaluations are done on a single H100 GPU with batch size one. For parallel and ADJUST methods, we vary the number of tokens sampled per diffusion denoising step to control the token throughput. Tokens sampled per denoising step varies from one to six. For DPD, we vary the confidence threshold for accepting tokens from 1.0 to 0.4. Recall that DPD is a greedy (temperature=0) sampling based method and hence we omit unconditional generation results for DPD, which requires sampling.

Results are encapsulated in the caption of Fig 3. We would like to emphasize that ADJUST for Dream-Base is trained in a downstream agnostic manner. The discrepancy between tokens/sec between the unconditional generation and GSM8k plots can be attributed to the longer eight-shot prompt used for GSM8k evaluation.

4.5 ParallelBench

ParallelBench [7] is a recent benchmark that quantifies the quality degradation under parallel decoding with diffusion LMs. We here report numbers for a representative task word-to-sentence using Dream-Base-7B as the base diffusion model in Table 3. The task prompt asks the model to generate a coherent sentence with a set of given words present in the sentence. A generation is valid if it's both coherent (grammatically) and includes all the specified words. We report the fraction of valid generation for three task difficulty levels (easy (E), medium (M) and hard (H)). We kindly refer the reader to App. Table 5) for results on the other tasks in parallelbench.

	W2S(E)	W2S(M)	W2S(H)
True Joint	<u>82.0</u>	**87.0**	**69.0**
Parallel (K=4)	81.0	77.0	63.0
ADJUST (K=4)	**84.0**	<u>85.0</u>	**69.0**

Table 3: In this table, we report numbers on the word-to-sentence (W2S) task of ParallelBench [7].

5 Conclusion, Limitations and Future Work

Conclusion. We study the problem of jointly sampling multiple tokens in each diffusion denoising step for masked diffusion LMs. Our main contribution was the formalization of our approximate joint sampler ADJUST, along with proposing a new training methodology based on sampling trajectory. For **future work**, we hope to design systems which can help ADJUST realize it's gains on token per pass to similar gains on tokens per second. **Limitations.** Since the only open-source diffusion LLM exist at 7B scale, we don't have ablations on base model size. For the same reasons, we only consider masked diffusion model as all open-source diffusion build on this paradigm.

References

[1] Marianne Arriola, Yair Schiff, Hao Phung, Aaron Gokaslan, and Volodymyr Kuleshov. Encoder-decoder diffusion language models for efficient training and inference. *arXiv preprint arXiv:2510.22852*, 2025.

[2] Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne Van Den Berg. Structured denoising diffusion models in discrete state-spaces. *Advances in neural information processing systems*, 34:17981–17993, 2021.

[3] Andrew Campbell, Valentin De Bortoli, Jiaxin Shi, and Arnaud Doucet. Self-speculative masked diffusions. *arXiv preprint arXiv:2510.03929*, 2025.

[4] Gabe Guo and Stefano Ermon. Reviving any-subset autoregressive models with principled parallel sampling and speculative decoding. *arXiv preprint arXiv:2504.20456*, 2025.

[5] Siming Huang, Tianhao Cheng, Jason Klein Liu, Jiaran Hao, Liuyihan Song, Yang Xu, J. Yang, J. H. Liu, Chenchen Zhang, Linzheng Chai, Ruifeng Yuan, Zhaoxiang Zhang, Jie Fu, Qian Liu, Ge Zhang, Zili Wang, Yuan Qi, Yinghui Xu, and Wei Chu. Opencoder: The open cookbook for top-tier code large language models. 2024.

[6] Daniel Israel, Guy Van den Broeck, and Aditya Grover. Accelerating diffusion llms via adaptive parallel decoding. *arXiv preprint arXiv:2506.00413*, 2025.

[7] Wonjun Kang, Kevin Galim, Seunghyuk Oh, Minjae Lee, Yuchen Zeng, Shuibai Zhang, Coleman Hooper, Yuezhou Hu, Hyung Il Koo, Nam Ik Cho, et al. Parallelbench: Understanding the trade-offs of parallel decoding in diffusion llms. *arXiv preprint arXiv:2510.04767*, 2025.

[8] Samar Khanna, Siddhant Kharbanda, Shufan Li, Harshit Varma, Eric Wang, Sawyer Birnbaum, Ziyang Luo, Yanis Miraoui, Akash Palrecha, Stefano Ermon, et al. Mercury: Ultra-fast language models based on diffusion. *arXiv preprint arXiv:2506.17298*, 2025.

[9] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-3: Scaling up inference acceleration of large language models via training-time test. *arXiv preprint arXiv:2503.01840*, 2025.

[10] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[11] Anji Liu, Oliver Broadrick, Mathias Niepert, and Guy Van den Broeck. Discrete copula diffusion. *arXiv preprint arXiv:2410.01949*, 2024.

[12] Lang Liu, Krishna Pillutla, Sean Welleck, Sewoong Oh, Yejin Choi, and Zaid Harchaoui. Divergence Frontiers for Generative Models: Sample Complexity, Quantization Effects, and Frontier Integrals. In *NeurIPS*, 2021.

[13] Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. *arXiv preprint arXiv:2310.16834*, 2023.

[14] Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. *arXiv preprint arXiv:2502.09992*, 2025.

[15] Litu Rout, Constantine Caramanis, and Sanjay Shakkottai. Anchored diffusion language model. *arXiv preprint arXiv:2505.18456*, 2025.

[16] Subham Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. *Advances in Neural Information Processing Systems*, 37:130136–130184, 2024.

[17] Jiaxin Shi, Kehang Han, Zhe Wang, Arnaud Doucet, and Michalis Titsias. Simplified and generalized masked diffusion for discrete data. *Advances in neural information processing systems*, 37:103131–103167, 2024.

[18] Yuxuan Song, Zheng Zhang, Cheng Luo, Pengyang Gao, Fan Xia, Hao Luo, Zheng Li, Yuehang Yang, Hongli Yu, Xingwei Qu, et al. Seed diffusion: A large-scale diffusion language model with high-speed inference. *arXiv preprint arXiv:2508.02193*, 2025.

[19] Chengyue Wu, Hao Zhang, Shuchen Xue, Zhijian Liu, Shizhe Diao, Ligeng Zhu, Ping Luo, Song Han, and Enze Xie. Fast-dllm: Training-free acceleration of diffusion llm by enabling kv cache and parallel decoding. *arXiv preprint arXiv:2505.22618*, 2025.

[20] Minkai Xu, Tomas Geffner, Karsten Kreis, Weili Nie, Yilun Xu, Jure Leskovec, Stefano Ermon, and Arash Vahdat. Energy-based diffusion language models for text generation. *arXiv preprint arXiv:2410.21357*, 2024.

[21] Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Dream 7b: Diffusion large language models. *arXiv preprint arXiv:2508.15487*, 2025.

[22] Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. Metamath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284*, 2023.

[23] Kaiwen Zheng, Yongxin Chen, Hanzi Mao, Ming-Yu Liu, Jun Zhu, and Qinsheng Zhang. Masked diffusion models are secretly time-agnostic masked models and exploit inaccurate categorical sampling. *arXiv preprint arXiv:2409.02908*, 2024.

A Additional Discussions

A.1 Hyper-parameter selection

EDLM We use Qwen-2.5-0.5B to evaluate generative perplexity of sampled strings and sample two string completions. For EDLM we choose just two samples as this was shown to be optimal number presented in the EDLM paper [20]. We ablated with 4 choices as well but got similar results. Hence we just consider sampling 2 options. Another consideration for EDLM is the choice of the auto-regressive model. We take the model to be Qwen2.5-0.5B. We also tried Qwen2.5-7B but that also didn't help improve the performance much. The numbers are reported as EDLM (Big) in Table 6.

B Additional Baselines

B.1 Discrete Copula Diffusion

DCD uses an auto-regressive model of the same size and distribution as the base diffusion model to generate a complete unmasked token string. An auto-regressive generation strategy uses $|\mathcal{M}(\boldsymbol{x})|$ forward passes of the AR model (of the same size as the diffusion model). We can alternatively use $|\mathcal{M}(\boldsymbol{x})|$ forward passes of the diffusion model (with $K = 1$) to directly sample from the true joint (without using the AR model). DCD **(author?)** [11] aims to construct a joint distribution over the whole token string instead of just tokens to be unmasked. Since they have a joint over all the masked positions, their method works in a regime of aggressive unmasking. ADJUST does auto-regressive generation over only the tokens to be unmasked. Hence, our method (intuitively) predicts a joint over positions to be unmasked and not the whole sentence. This makes our method much more computationally friendly then DCD.

B.2 Adaptive parallel decoding

(APD) [6]: is a specialized generation technique restricted to left-to-right decoding. At each denoising step APD samples a complete unmasked token string (in parallel). It then uses an auto-regressive evaluator (model) to adaptively selects a prefix string to commit and unmask. Because APD generates variable number of tokens in each denoising step and is strictly left-to-right.

We first note that APD is a specialized left-to-right constrained decoding algorithm suited for reasoning heavy tasks evaluated at low temperatures. For e.g., for Dream-7B-Instruct, using APD we can generate (on an average) upto five tokens per denoising iteration for low temperature (=0.1) sampling on GSM8k prompts with negligible loss in performance. Here we investigate the performance of

methods for open-ended tasks like unconditional generation and high temperature GSM8k sampling. We acknowledge the synthetic nature of high temperature GSM8k sampling and emphasize that these numbers are for completeness.

Results are encapsulated in caption of Fig 4. We highlight that we use left-to-right decoding for APD and use base model's unmasking logic for parallel and ADJUST. The results show that ADJUST is applicable across tasks and sampling temperatures where APD fails.

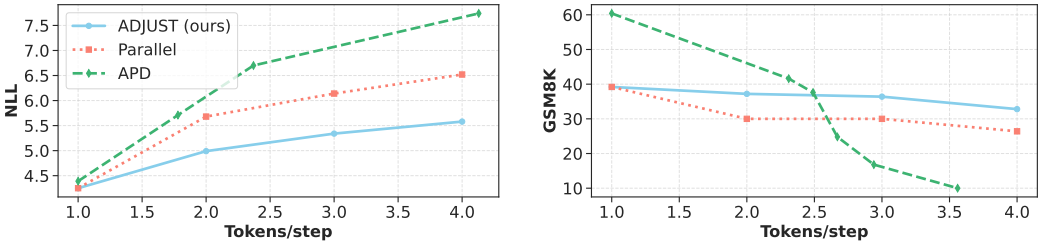

Figure 4: In this figure we compare adaptive parallel decoding (working left-to-right) to naive parallel decoding and ADJUST. Sampling $K > 1$ tokens per diffusion step, for left-to-right decoding shows a steeper increase in NLL (left) than using the base model's unmasking logic. Hence APD generation is not suited for such open-ended tasks and under-performs. The right figure shows GSM8k evaluation for high temperature sampling (temperature=1.0) for Dream-Instruct model. Left-to-right generation is naturally good for math reasoning tasks, as shown by higher accuracy under APD when compared to naive/joint for $K = 1$ tokens per step. APD leads to a quicker decrease in accuracy when compared to baseline and our method.

C Additional Experiments from omitted from the main text

C.1 Complete NLL and MAUVE numbers

See Table 4 for all numbers

		K=1		**K=2**		**K=3**		**K=4**	
		NLL	Mauve	NLL	Mauve	NLL	Mauve	NLL	Mauve
T=0.6	Parallel	1.11	1.00	1.61	0.89	2.18	0.57	2.21	0.29
	EDLM	**0.83**	0.98	1.33	0.94	1.85	0.54	1.89	0.31
	ADJUST **(ours)**	1.11	1.00	**1.23**	**0.97**	**1.54**	**0.82**	**1.64**	**0.79**
T=1.0	Parallel	4.25	1.00	5.68	0.76	6.14	0.48	6.52	0.31
	EDLM	**3.75**	0.96	5.26	0.83	5.76	0.58	6.18	0.39
	ADJUST **(ours)**	4.25	1.00	**4.99**	**0.96**	**5.34**	**0.90**	**5.58**	**0.87**
		K=5		**K=6**		**K=7**		**K=8**	
		NLL	Mauve	NLL	Mauve	NLL	Mauve	NLL	Mauve
T=0.6	Parallel	2.00	0.12	2.11	0.07	2.19	0.05	2.21	0.02
	EDLM	1.87	0.14	1.95	0.07	2.03	0.05	2.05	0.03
	ADJUST **(ours)**	**1.47**	**0.79**	**1.54**	**0.77**	**1.60**	**0.76**	**1.66**	**0.67**
T=1.0	Parallel	6.78	0.22	6.80	0.24	6.89	0.17	7.10	0.19
	EDLM	6.19	0.27	6.26	0.26	6.35	0.21	6.56	0.16
	ADJUST **(ours)**	**5.67**	**0.87**	**5.76**	**0.85**	**5.83**	**0.85**	**5.91**	**0.84**

Table 4: In this table we compare the NLL and the MAUVE scores for three different methods : Marginal, Joint and Energy-based (EDLM). Across different number of tokens K sampled per step, and different sampling temperature, joint sampling consistently outperforms both of the baselines, parallel sampling and EDLM.

C.2 Complete ParallelBench Numbbers

See Table 5

Model	W2S(E)	W2S(M)	W2S(H)	Copy	Ins(idx)	Ins(rand)	Rm(idx)
Dream (x1)	82.0	87.0	69.0	73.0	30.0	42.0	72.0
Parallel (x4)	81.0	77.0	63.0	55.0	12.0	13.0	57.0
ADJUST (x4)	84.0	85.0	69.0	63.0	21.0	20.0	62.0
Model	Rm(rand)	Rpl(idx)	Rpl(rand)	Reverse	Shuffle	Sort	
Dream (x1)	62.0	51.0	52.0	68.0	62.0	21.0	
Parallel (x4)	36.0	35.0	1.0	27.0	24.0	13.0	
ADJUST (x4)	39.0	46.0	3.0	39.0	14.0	16.0	

Table 5: Evaluation on ParallelBench using Dream-Base-7B model

C.3 EDLM Experiments

	T=0.6			**T=1.0**		
	Spec=1	Spec=2	Spec=3	Spec=1	Spec=2	Spec=3
Tok/sec	35.57	52.87	71.00	35.56	52.91	70.97

	T=0.6			**T=1.0**		
	Spec=1	Spec=2	Spec=3	Spec=1	Spec=2	Spec=3
EDLM (Small)	1.33	1.85	1.89	5.26	5.76	6.18
EDLM (Big)	1.61	2.08	2.08	5.27	5.74	6.18

Table 6: We report the runtime of the EDLM algorithm (with Qwen-2.5-0.5B) for experimental setup in Table 1 in Tokens/sec. These numbers are comparable to those in figure 1. In this table we report the negative log-likelihood numbers on decoding with EDLM. We fix the generation length to be 128 tokens and nucleus sampling to be 0.95, and temperature to be 1.0. Here we use Qwen2.5-7B as the verifier for EDLM (Big), i.e., the same model which is used to evaluate responses, while using Qwen2.5-0.5B as the verifier for EDLM (Small). We can see identical results because at each position we sample only two options both of which are identically rated by both the small and the big models

C.4 Architecture

See Fig 5.

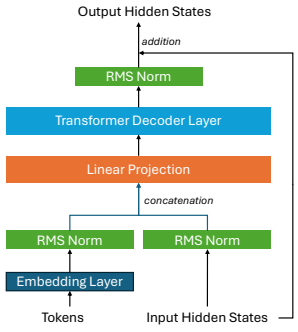

Figure 5: This figure shows the architecture used for our draft model

D Additional Experimental Details

D.1 Sampling Parameters

For downstream evaluation, we take parameters from the official Dream repository[3]

Pretrained-only (Dream-Base/Llada-Base) For NLL and MAUVE evaluation we use temperature in $\{1.0, 0.6\}$ and topp of 1.0 with generation length of 128. We summarize them here for completeness. For GSM8K evaluation we use temperature of 0.0 and 256 generation length with 8 shot examples in context. For MBPP we use 0.2 temperature, topp of 1.0 with generation length of 512 and num few shot of 3 examples.

Dream-7B-Instruct For GSM8K we use 0.1 temperature, 0.9 topp, and generation length of 256. Also we use the chat template for evaluation. For Minerva-math we use 4 few shot, 512 generation length and 0.1 temperature and 0.9 topp. Our evaluation code for Minerva relies on an older version from lm-evaluation-harness that doesn't utilize "math_verify" and hence the numbers might be under-reported here.

Dream-7B-Coder For HumanEval we use 0.2 temperature, 0.9 topp, and generation length of 512 and four few-shot examples. Similarly for MBPP we use the same sampling parameters

E Unmasking Logic

In our discussion till now, we assumed the list of positions to be generated σ to be given to us. For the current state-of-the-art diffusion LLMs, positions to be generated next are decided by an operator which is a function of the logits, denoted as TOP. Base diffusion language models specify their criterion to decide the most suitable position to generate next. For example, Dream models [21] select the position with the least entropy to generate next, Llada [14] unmasks position with the highest confidence. ADJUST is agnostic to the choice of the unmasking logic and uses the base model's unmasking logic for generation. A subtle difference to note is that naive parallel sampling of K tokens calls TOP on the same output embeddings $f(\boldsymbol{x})$, K times, yielding top K positions with (say) least entropy in $\boldsymbol{W}f$. ADJUST uses the updated logits to get the index of the next position, $\sigma(k) = \text{TOP}\{\boldsymbol{W}\boldsymbol{h}_k\}$, which we qualitatively observe are often different than using the base model's logits $\boldsymbol{W}f$.

[3] `https://github.com/DreamLM/Dream`

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: All claims in the abstract are accurately represented in the paper

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We have Sec 5 in the paper which discusses our limitations

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

Justification: We don't have any theoretical results.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Yes, all details are shared. Work is reproducible.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [No]

Justification: The dataset is open-source. We use open-source repository for experimentation and can be done easily.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Experimentation details are there in Sec 4.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: The standard errors are small for all the numbers reported. For ease of exposition we have not included them.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).

- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Yes, compute requirements are in the main paper Sec 4.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We conform to the code of ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: Our work advances language models and there are no additional malicious use-cases to the best of our knowledge, beyond using better language models for harm.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [No]

    Justification: No such risks

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [N/A]

    Justification: All the data used is open-source and appropriate citations are given.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: No new assets

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: No crowdsourced research

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: No such research

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: LLMs are not used.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.