ENABLING APPROXIMATE JOINT SAMPLING IN DIFFUSION LMS

Anonymous authors
Paper under double-blind review

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) and instruction-tuned (Dream-7B-Instruct) models on language modeling and math & coding tasks. When four tokens are unmasked for each full-model denoising step, our sampling algorithm achieves a MAUVE score of 0.87 (vs marginal baseline of 0.31) with respect to the true joint distribution.

1 INTRODUCTION

Masked diffusion language models Sahoo et al. (2024); Austin et al. (2021); Lou et al. (2023) 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 Sahoo et al. (2024). 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 Ye et al. (2025); Nie et al. (2025) 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 Israel et al. (2025).

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. We are not the first to make the observation that parallel sampling is akin to product-marginal sampling, but we are the first to do something about it. 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 consists of a single trainable transformer block layer on top of an existing (frozen) diffusion model. Each denoising step with ADJUST corresponds to a single forward pass of the base diffusion model followed up by $K - 1$ iterative forward passes of the sampler layer; each of these K unmasks an additional token. We carefully design the specific architecture of ADJUST, as well as specialized training data, loss function and training algorithm for it; the objective of all of these is for the output of ADJUST to mimic true (slow) joint sampling as closely as possible.

We note that while our method is inspired by speculative decoding papers, is *not* speculative decoding; because, there is no "drafting and verification". Indeed, the final generated strings from our method represent a different distribution than the base model we start with; different both from what the base model would have generated in parallel-unmasking mode and also different from what it would have generated in slow single-token-unmasking mode. In contrast, in speculative decoding the emphasis is on faster generation while mirroring the distribution of the base model. We also note that our method does not need access to any outside "similarly distributed" or "checker" auto-regressive model.

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 forward pass. Evidence of the value of joint sampling can be seen in existing work, where it shows up as increased accuracy with increased forward passes. 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 3.3 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, ADJUST increases GSM8k accuracy by 16 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.

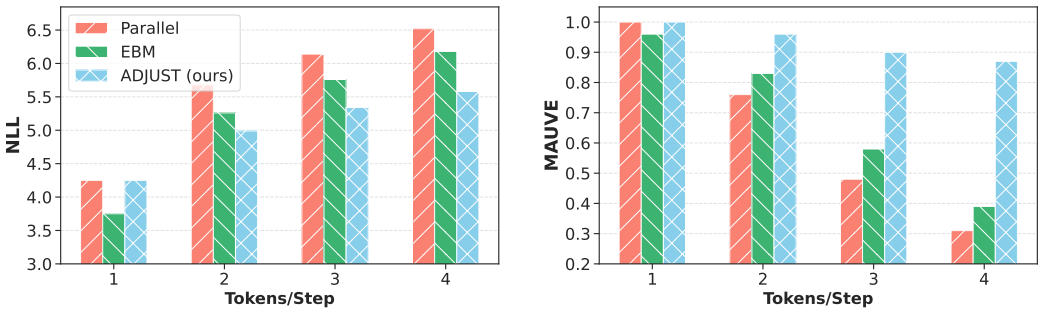

Figure 1: In this figure, we use Dream-7B-Base to generate a token string of length 128 tokens starting from an all masked string (kindly refer to Sec 4.2 for details). We report the negative log-likelihood (NLL) and the MAUVE score for the generated strings. We vary the number of tokens sampled per forward pass of the diffusion model (denoising step) from one to four. We observe an increase in NLL and a decrease in the MAUVE score as more tokens are generated in parallel. We argue this is because generating multiple tokens in parallel samples from a distribution different from the true joint distribution. For a given number of tokens generated per step, our joint sampler ADJUST achieves the best NLL and MAUVE (as compared to baseline methods like naive parallel sampling, and energy-based models).

2 Related Work

Diffusion language models Discrete diffusion language models (LMs) Sahoo et al. (2024); Shi et al. (2024); Lou et al. (2023) serve as a promising alternative to auto-regressive (AR) LMs. Large diffusion LMs Ye et al. (2025); Nie et al. (2025) competing with the AR counterparts are based on an absorbing-state Markov process, which noises tokens into a "masked" state. Masked diffusion language models can also be considered as mask denoisers Zheng et al. (2024). Discrete Copula Diffusion Liu et al. (2024b) uses both AR and diffusion models in tandem to get final logits which have the best of both worlds (future conditioned and joint probabilities). One line of work focuses on how to select the next token to unmask and propose trainable planners for achieving the best downstream performance Peng et al. (2025); Liu et al. (2024c). Apart from diffusion language models, we also mention other work closely related to this paradigm such as any-order auto-regressive models Shih et al. (2022), among others Sahoo et al. (2025); Chao et al. (2025)

Efficient decoding for language models Prior work has proposed techniques for speeding up diffusion language models by introducing KV cache and adaptive decoding of confident tokens Wu et al. (2025); Israel et al. (2025). These works leverage the confidence score of the diffusion model or an off-the-shelf AR model to decide how many tokens to commit often speeding up inference. Speculative decoding is a popular paradigm for provably accelerating auto-regressive models Leviathan et al. (2023). Prior work on this has developed drafter techniques which heavily inspire our method Li et al. (2025; 2024); Liu et al. (2024a); Zhang et al. (2024). Prior work also introduces unrolling the draft model during training (to better align test and train) which we use in our work Li et al. (2025); Liu et al. (2024a); Zhang et al. (2024). Recent work also proposes speculative decoding for any-order models Guo & Ermon (2025).

3 Method

3.1 Problem Setup

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 the 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 unmasked tokens in $\mathcal{V}/\text{M}$. 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.

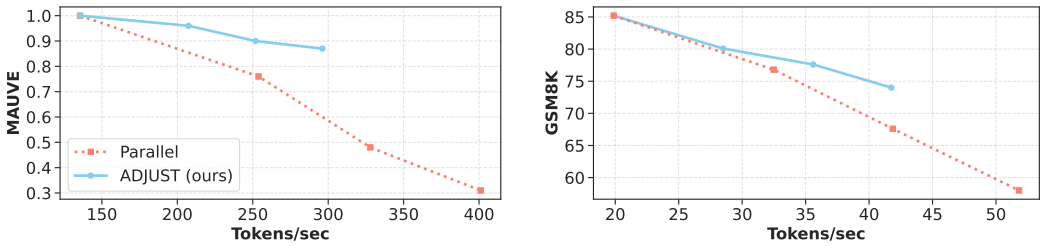

Figure 2: In this figure we report runtime (tokens produced per second) with the corresponding MAUVE and GSM8k scores. We vary the number of tokens sampled for a single diffusion denoising step to vary the token throughput (from one to four tokens per step, each corresponding to a point on the curve). Increasing the number of tokens sampled per step leads to a decrease in both MAUVE and GSM8k performance, as expected. We show that our joint sampler leads to only slight reduction in throughput compared to the naive parallel sampling. Importantly, for a given target throughput, sampling using our joint approximation outperforms using naive parallel sampling. For example, for parallel decoding of four tokens in each diffusion pass, our joint approximation is only 20-25% slower than parallel decoding while being 16 % more accurate on GSM8K and 0.5 points higher on MAUVE.

In this paper we only consider base models that are 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 $\phi \to \boldsymbol{x}_{n_1} \to \boldsymbol{x}_{n_2} \to \ldots \to \boldsymbol{x}_L$, where $\phi = \boldsymbol{x}_0$ is the all-mask sequence and $\boldsymbol{x}_L$ is the completely unmasked string. In this paper we will build a joint sampler for an existing, frozen base diffusion model which we will refer to by the pair $(f, \boldsymbol{W})$. Here $f : \mathcal{V}^L \to \mathbb{R}^{d \times L}$[1] refers to the map 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 i^{th} position. $\boldsymbol{W} \in \mathbb{R}^{\mathcal{V} \times d}$ is the language model head that maps an embedding to logits. Note that **diffusion models only output per-token marginals**; that is for each position i 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.

We now build our intuitive notion of the ***true underlying joint distribution*** p_* represented by the base diffusion model. Recall that in any single forward pass, the model outputs L single-token probability distributions $p^i(\cdot|\boldsymbol{x})$, for every position $i = 1, \ldots, L$. 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 $[L]$):

$$\begin{aligned} p_*(\boldsymbol{x}) &= \prod_{k=1}^{L} p_*(x^{\sigma(k)} \,|\, \phi \oplus x^{\sigma(1)} \oplus \ldots \oplus x^{\sigma(k-1)}) \\ &= \prod_{k=1}^{L} p^{\sigma(k)}(x^{\sigma(k)} \,|\, \phi \oplus x^{\sigma(1)} \oplus \ldots \oplus x^{\sigma(k-1)}) \end{aligned}$$

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

The 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

[1] we assume that the time information is implicit in the masked input sequence Zheng et al. (2024)

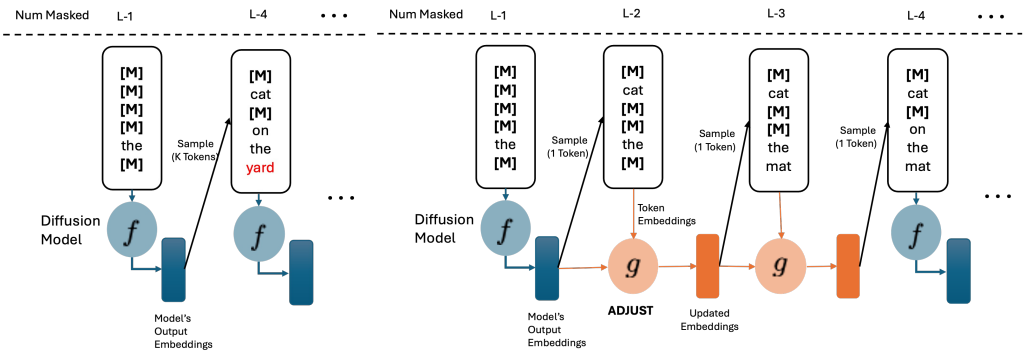

Figure 3: In this figure we illustrate our method ADJUST. The sub-figure on the left shows naive parallel sampling, while the sub-figure on the right shows our method ADJUST. In this example, each denoising step generates three tokens. The underlying diffusion model has support on only two distinct sentences, "The cat sat on the mat" and "The dog ran in the yard". Sampling tokens in parallel (left), generates an incoherent sentence (w.r.t. the underlying distribution of diffusion model). Diffusion model's support does not contain a sentence which mentions a "cat", "on the yard". On the other hand, our method ADJUST (right) conditions each token sample on the previously sampled tokens through a light-weight network, shown in the figure as g. Note that we utilize the same number of forward passes of the diffusion model f as the naive parallel sampler (once every three generated tokens).

joint distribution satisfies

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

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 approximate joint sampler

We develop a new method to generate – i.e. sample and unmask – K tokens with one forward pass of the base diffusion model. We do this by sampling one token at a time from a **small extra draft model** (which we term ADJUST). Crucially, *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 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 3.

ADJUST The ADJUST model is represented as the pair $(g, \boldsymbol{W})$, similar to how the base diffusion model is $(f, \boldsymbol{W})$. The function $g : \mathbb{R}^{d\times L} \times \mathcal{V}^L \to \mathbb{R}^{d\times L}$ takes two inputs: $\boldsymbol{h} \in \mathbb{R}^{d\times L}$, which is a set of "most current" embeddings (elaborated on later), as well as the currently unmasked string (say $\tilde{\boldsymbol{x}}$). g represents the output embeddings of ADJUST. Let q denote the output marginal probabilities of ADJUST. Then we can denote the probabilities as :

$$q\left(\cdot \mid \boldsymbol{h}, \tilde{\boldsymbol{x}}\right) = \text{softmax}\left[\boldsymbol{W} g(\boldsymbol{h}, \tilde{\boldsymbol{x}})\right]$$

where $\boldsymbol{h}$ and $\tilde{\boldsymbol{x}}$ are the most current embeddings and token strings respectively. 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.

Let us now walk through how ADJUST sequentially generates tokens; let $\sigma(1), \ldots, \sigma(K)$ be the set of positions it will unmask. The first token $\tilde{x}^{\sigma(1)}$ can be unmasked directly using the base model forward pass $p^{\sigma(1)}(\cdot \mid \boldsymbol{x})$ itself. After sampling the first token, the most current token string gets updated to $\boldsymbol{x} \oplus \tilde{x}^{\sigma(1)}$, while our most current embedding is $f(\boldsymbol{x})$. From the second token onward our

Algorithm 1 ADJUST: Approximate Joint Sampling

Input: Prompt $\boldsymbol{x}$, frozen discrete diffusion model $f, \boldsymbol{W}$ (where $\boldsymbol{W}$ is the language model 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**
  $\boldsymbol{h}_1 \leftarrow f(\boldsymbol{x})$ ▷ Compute output embeddings of the base model
  for $k = 1$ to K **do**
    $i \leftarrow \text{TOP}\{\boldsymbol{W}\boldsymbol{h}_k^j : j \in \mathcal{M}(\boldsymbol{x})\}$ ▷ Select index to unmask
    $\tilde{x}^i \sim \text{softmax}(\boldsymbol{W}\boldsymbol{h}_k^i)$
    $\boldsymbol{x} \leftarrow \boldsymbol{x} \oplus \tilde{x}^i$ ▷ Get updated token string
    $\boldsymbol{h}_{k+1} \leftarrow g(\boldsymbol{h}_k, \boldsymbol{x})$ ▷ Update output embeddings using ADJUST
  end for
end while
return $\boldsymbol{x}$

method ADJUST kicks-in. The most current embeddings are first updated (with the most current string) using ADJUST as :

$$\boldsymbol{h}_2 = g\left(f(\boldsymbol{x}), \boldsymbol{x} \oplus \tilde{x}^{\sigma(1)}\right)$$

and then the second token $\tilde{x}^{\sigma(2)}$ is sampled using the logits $\boldsymbol{W}\boldsymbol{h}_2$. This means that

$$p_{\text{ADJUST}}(\tilde{x}^{\sigma(1)}, \tilde{x}^{\sigma(2)} \mid \boldsymbol{x}) \;=\; p^{\sigma(1)}\left(\tilde{x}^{\sigma(1)} \mid \boldsymbol{x}\right)\; q^{\sigma(2)}\left(\tilde{x}^{\sigma(2)} \mid f(\boldsymbol{x}), \boldsymbol{x} \oplus \tilde{x}^{\sigma(1)}\right)$$

Now for sampling of the third token, our most current embeddings are $\boldsymbol{h}_2$ and most current token string is $\boldsymbol{x} \oplus \tilde{x}^{\sigma(1)} \oplus \tilde{x}^{\sigma(2)}$. These are used as input during the next iteration of g. Hence, letting $\boldsymbol{h}_1 = f(\boldsymbol{x})$, we can define the recursion for output embeddings of g as

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

and the final K-step probability as :

$$\begin{aligned} &p_{\text{ADJUST}}(\tilde{x}^{\sigma(1)}, \ldots, \tilde{x}^{\sigma(K)} \mid \boldsymbol{x}) \\ &= p^{\sigma(1)}\left(\tilde{x}^{\sigma(1)} \mid \boldsymbol{x}\right) \prod_{k=2}^{K} q^{\sigma(k)}\left(\tilde{x}^{\sigma(k)} \mid \boldsymbol{h}_{k-1}\,,\, \boldsymbol{x} \oplus \tilde{x}^{\sigma(1)} \oplus \ldots \oplus \tilde{x}^{\sigma(k-1)}\right) \end{aligned}$$

Note that for sampling K tokens, we do a single forward pass through the base diffusion model, for computing $\boldsymbol{h}_1 = f(\boldsymbol{x})$ and $K-1$ forward passes through our ADJUST g for computing all subsequent $\boldsymbol{h}_k$. We present our joint sampling algorithm in Alg 1.

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.

Is this speculative decoding? We clarify that our method is NOT a speculative decoding Leviathan et al. (2023) method. Speculative decoding refers to a "guess and verify" paradigm for accelerating inference through auto-regressive (AR) language models. It is specifically useful for AR models as the verification (NLL computation) of a guessed string (a single forward pass through the AR model) is cheaper than auto-regressive generation (K forward passes). Since diffusion models do not provide an efficient way to verify tokens which is cheaper than generating the tokens themselves, speculative decoding techniques are not applicable for diffusion models. We highlight two other differences between ADJUST and speculative decoding. Firstly, as shown above, we always accept all the tokens generated by ADJUST (there is no verification). Recall that diffusion models, through a single forward pass, give us a marginal distribution (*prior belief*) for each masked position. Our formulation considers conditioning this prior belief on the sampled tokens. This is in contrast to drafters used for AR models in speculative decoding, where the goal is to *speculate* future tokens from scratch (no prior belief is given by the base model).

Algorithm 2 ADJUST Training

Input: Distribution $\mathcal{D}$ of input strings (each of which contains masked and unmasked positions), frozen discrete diffusion model $f, \boldsymbol{W}$ (where $\boldsymbol{W}$ is the language model head), number K of tokens to generate in each step, base model's unmasking logic TOP
repeat
  $\boldsymbol{x} \sim \mathcal{D}$
  $\boldsymbol{x} \leftarrow$ 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 $\boldsymbol{x}$
  $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_K \leftarrow f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_K)$ ▷ Compute output embeddings
  $\hat{\boldsymbol{h}}_0 \leftarrow f(\boldsymbol{x})$
  for $k = 1$ to K **do**
    $\hat{\boldsymbol{h}}_k \leftarrow g_\theta(\hat{\boldsymbol{h}}_{k-1}, \boldsymbol{x}_k)$
  end for
  $\mathcal{L}(\theta) \leftarrow -\sum_k D_{\mathrm{KL}}(\boldsymbol{W}\hat{\boldsymbol{h}}_k, \boldsymbol{W}\boldsymbol{h}_k)$
  Take gradient step on $\nabla_\theta \mathcal{L}(\theta)$
until converged
return g_θ

3.3 Training Methodology

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 Ye et al. (2025) select the position with the least entropy to generate next, Llada Nie et al. (2025) 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 $\boldsymbol{W}f$.

Training Loss We want our joint approximation p_{ADJUST} to mimic the true joint distribution p_*. We do this by minimizing the KL divergence between the true joint and our approximation. Our training loss is hence a sum of KL divergence terms between $p^{\sigma(k)}\left(\cdot \mid \boldsymbol{x} \oplus \tilde{x}^{\sigma(1)} \oplus \ldots \oplus \tilde{x}^{\sigma(k-1)}\right)$ and $q^{\sigma(k)}\left(\cdot \mid \boldsymbol{h}_{k-1}, \boldsymbol{x} \oplus \tilde{x}^{\sigma(1)} \oplus \ldots \oplus \tilde{x}^{\sigma(k-1)}\right)$ for all values of $k \geq 2$. We still haven't defined how $\boldsymbol{x}$ is generated or how we get σ or $\boldsymbol{h}$ for computing the loss. We elaborate on these next.

Firstly, we describe how we select $\boldsymbol{x}$ for training. 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 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 $\boldsymbol{x}$. 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 $g\left(\boldsymbol{h}_{k-1}, \boldsymbol{x} \oplus \tilde{x}^{\sigma(1)} \oplus \ldots \oplus \tilde{x}^{\sigma(k-1)}\right)$. We use the recursive formulation of g during training as well to compute $\boldsymbol{h}_k$. Since k is small (≈ 4), this is not computationally expensive. The "unrolling" of g during training is considered helpful by prior work Liu et al. (2024a); Li et al. (2025) and hence we do not ablate on this choice. Putting these things together, we present our training algorithm in Alg 2.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

We consider two different state-of-the-art diffusion language models as base models for our empirical investigation. We consider a pretrained-only model (Dream-7B-Base) and an instruction-tuned model (Dream-7B-Instruct) Ye et al. (2025). Recall that, generating 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.

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* Xu et al. (2024): EBM 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. Intuitively, the selection operator of EBM biases the samples towards the distribution of the auto-regressive model and away from the diffusion model. **4)** *Adaptive parallel decoding* (APD) Israel et al. (2025): is a specialized generation technique restricted to left-to-right decoding. At each denoising step, similar to EBM, APD samples a complete unmasked token string (in parallel). It then uses an auto-regressive evaluator (model) to compute the generative perplexity of all prefixes of the unmasked string and adaptively selects a prefix to commit and unmask. APD generates variable number of tokens in each denoising step. We compare with APD 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 another baseline method *discrete copula diffusion* (DCD) Liu et al. (2024b) which we skip in our experimental setup. DCD uses an auto-regressive and diffusion model in tandem to get final output logits. Their formulation is orthogonal to our goal of mimic generation from the true joint distribution. (see Sec A.2 for additional discussion). We also highlight that all the baseline methods, assume an access to a "similarly" distributed auto-regressive model. This assumption though is valid for Dream models, can be restrictive. ADJUST does not rely on this assumption and works with any off-the-shelf diffusion model.

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). ADJUST assumes an initial distribution of prompts $\mathcal{D}$ (kindly refer Alg 2). For Dream-Base model 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-Base. For Dream-Instruct, we subsample 20K unique questions from MetamathQA and generate with max generation length of 256 tokens, with temperature of 0.1 and nucleus of 0.9. We choose the data-generation parameters to align with our evaluation setup (though our results show robustness to these).

4.2 UNCONDITIONAL GENERATION

We consider generation with the input prompt is just the beginning-of-sentence token, with a max generation length of 128. We call this task *unconditional generation*. We generate the output strings using our baseline methods and compare the 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

		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
	EBM	**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
	EBM	**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**

Table 1: In this table we compare the NLL and the MAUVE scores for three different methods : Marginal, Joint and Energy-based (EBM). Across different number of tokens K sampled per step, and different sampling temperature, joint sampling consistently outperforms both of the baselines, parallel sampling and EBM.

MAUVE score Liu et al. (2021) (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 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 $K = 1$, EBM has lower NLL compared to our method, while (seemingly counterintuitively) having lower MAUVE. This is an expected behavior, as EBM samples two different completions for each denoising step and chooses the completion with a lower NLL. The selection of lower NLL completion biases the EBM 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 EBM achieves lower NLL than parallel and ADJUST for $K = 1$. We omit discussion of APD here and kindly refer readers to Sec 4.4.

4.3 Downstream Evaluation

We evaluate different methods on downstream tasks in this section. Sampling from the true joint distribution, $K = 1$, leads to the best downstream accuracy. Hence 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 value of K. We use low temperature sampling (0.0-0.1) for generation. Exact sampling parameters are summarized in Sec D.1. We use lm-eval harness for evaluation. Due to computational constraints we limit our evaluation to 250 question from GSM8K & MBPP and 350 question (50 question per task) for Minerva math. Any mismatch between our numbers and the reported numbers (especially on Minerva-math) can be attributed to this artifact.

Results Recall that the ADJUST for the base model is trained on domain-agnostic data (generations starting from begin-of-sentence token). Hence these results 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, EBM leads to same performance as naive sampling. Hence we don't report those numbers in the table. ADJUST for instruct model is trained on MetamathQA prompts. Hence we evaluate on math domain downstream tasks (GSM8K and Minerva-math). Across varying number of tokens sampled K per denoising step, our joint sampling outperforms naive generation. We omit discussion of APD in this section and kindly refer readers to Sec 4.4.

Runtime We report the runtime numbers for naive sampling and our joint sampling in Fig 2. The left plot shows numbers for Dream-Base model on MAUVE metric, while the right plot shows numbers for Dream-Instruct on GSM8k. The numbers show tokens produced per second and show

		K=1		K=2		K=3		K=4	
		GSM	MBPP	GSM	MBPP	GSM	MBPP	GSM	MBPP
Base	Parallel	78.40	48.00	71.20	40.80	58.80	33.20	43.20	33.60
	ADJUST(**ours**)	78.40	48.00	75.60	44.80	70.00	44.40	64.00	33.60
		GSM	MATH	GSM	MATH	GSM	MATH	GSM	MATH
Inst	Parallel	85.20	32.57	76.80	32.29	67.60	20.86	58.00	15.14
	ADJUST(**ours**)	85.20	32.57	80.08	31.42	77.60	26.00	74.00	22.28

Table 2: 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.

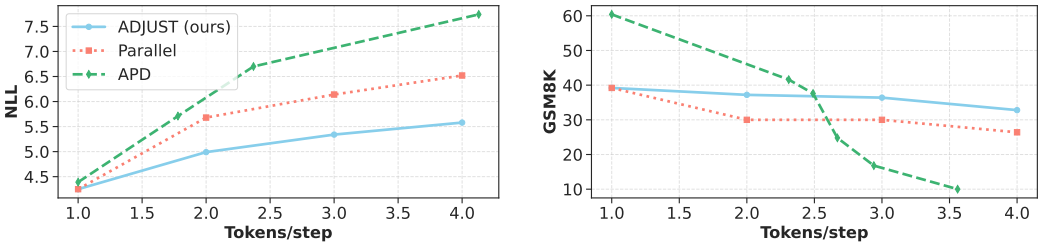

Figure 4: In this figure we compare adaptive parallel decoding (working left-to-right) to naive parallel decoding and ADJUST. Sampling $K > 1$ tokens per diffusion step, for left-to-right decoding shows a steeper increase in NLL (left) than using the base model's unmasking logic (see Table 4 for parallel generation, constrained to left-to-right decoding of $K > 1$ tokens per step). 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 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.

that our joint sampling method outperforms naive sampling for given throughput (tokens/sec) target. The discrepancy between tokens/sec between the two plots can be attributed to the longer sequence length and smaller batch-sizes used for GSM8k evaluation, as compared to unconditional generation. We report numbers for EBM on this figure, as EBM's throughput (in our setup) is order of magnitude smaller than both joint and naive sampling. We report tokens/sec performance for unconditional generation using EBM in Table 3 (≈ 50 toks/sec). We hypothesis that this is due to fact that EBM and APD both involve forward passes through multi layer (24 layers) models having thrice the number of parameters as our ADJUST model. Also note that ADJUST utilizes just a single transformer layer and hence is able to use parallel computing power of GPUs more effectively.

4.4 Comparison with adaptive parallel decoding

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 (see Table 5). 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 They 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.

REFERENCES

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.

Chen-Hao Chao, Wei-Fang Sun, Hanwen Liang, Chun-Yi Lee, and Rahul G Krishnan. Beyond masked and unmasked: Discrete diffusion models via partial masking. *arXiv preprint arXiv:2505.18495*, 2025.

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

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

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pp. 19274–19286. PMLR, 2023.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle: Speculative sampling requires rethinking feature uncertainty. *arXiv preprint arXiv:2401.15077*, 2024.

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.

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*, 2024a.

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

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.

Sulin Liu, Juno Nam, Andrew Campbell, Hannes Stärk, Yilun Xu, Tommi Jaakkola, and Rafael Gómez-Bombarelli. Think while you generate: Discrete diffusion with planned denoising. *arXiv preprint arXiv:2410.06264*, 2024c.

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

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.

Fred Zhangzhi Peng, Zachary Bezemek, Sawan Patel, Jarrid Rector-Brooks, Sherwood Yao, Avishek Joey Bose, Alexander Tong, and Pranam Chatterjee. Path planning for masked diffusion model sampling. *arXiv preprint arXiv:2502.03540*, 2025.

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.

Subham Sekhar Sahoo, Zhihan Yang, Yash Akhauri, Johnna Liu, Deepansha Singh, Zhoujun Cheng, Zhengzhong Liu, Eric Xing, John Thickstun, and Arash Vahdat. Esoteric language models. *arXiv preprint arXiv:2506.01928*, 2025.

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.

Andy Shih, Dorsa Sadigh, and Stefano Ermon. Training and inference on any-order autoregressive models the right way. *Advances in Neural Information Processing Systems*, 35:2762–2775, 2022.

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.

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.

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.

Lefan Zhang, Xiaodan Wang, Yanhua Huang, and Ruiwen Xu. Learning harmonized representations for speculative sampling. *arXiv preprint arXiv:2408.15766*, 2024.

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

EBM We use Qwen-2.5-0.5B to evaluate generative perplexity of sampled strings and sample two string completions. For EBM we choose just two samples as this was shown to be optimal number presented in the EBM paper Xu et al. (2024). We ablated with 4 choices as well but got similar results. Hence we just consider sampling 2 options. Another consideration for EBM 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 EBM (Big) in Table 3.

A.2 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 Liu et al. (2024b) 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 Additional Experiments

You may include other additional sections here.

B.1 EBM 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
EBM (Small)	1.33	1.85	1.89	5.26	5.76	6.18
EBM (Big)	1.61	2.08	2.08	5.27	5.74	6.18

Table 3: We report the runtime of the EBM 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 EBM. 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 EBM (Big), i.e., the same model which is used to evaluate responses, while using Qwen2.5-0.5B as the verifier for EBM (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

B.2 APD Experiments

B.2.1 NLL Experiments

We first argue that left-to-right decoding is not the right setting for generative perplexity. This can be seen as decoding multiple tokens in parallel for left-to-right decoding leads to worse negative log-

likelihood (as measured by Qwen2.5-7B) than standard entropy based unmasking used by Dream. This is shown by the top table in Table 4. The bottom table shows when we use adaptive parallel decoding for unmasking. Using APD helps increase toks/step while decreasing NLL, but it still doesn't outperform our joint sampler (or even marginal sampler) in NLL v/s Toks/Step plot (see Fig 4)

	Spec=0	Spec=1	Spec=2	Spec=3
Marginal	4.39	7.94	8.28	8.53

	R=0.3			**R=0.5**		
	NLL	Tok/step	Tok/sec	NLL	Tok/step	Tok/sec
Marg + APD(U)	5.71	1.78	14.55	6.70	2.37	18.77
	R=0.7			**R=0.9**		
	NLL	Tok/step	Tok/sec	NLL	Tok/step	Tok/sec
Marg + APD(U)	7.74	4.13	31.31	8.51	10.54	76.92

Table 4: In this table we report the negative log-likelihood numbers on left-to-right generation. 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 Qwen2.5-0.5B as the verifier for APD.

B.2.2 GSM8K Experiments

We here recreate APD algorithm (Table 5) number for Dream-7B-Base model denoted by Mar+APD (U). The results are encapsulated in the caption.

		R=0.3		**R=0.5**		**R=0.7**		**R=0.9**	
		Acc	T/s	Acc	T/s	Acc	T/s	Acc	T/s
Base	Mar + APD (U)	70.80	7.25	70.80	7.98	67.20	8.30	45.60	11.62
	Mar + APD	71.20	3.29	72.40	3.39	71.20	3.44	65.20	3.65
	Joint + APD	72.40	3.34	72.00	3.44	68.80	3.50	63.60	3.70

Table 5: In this table we adaptively decide the number of tokens to speculate by using the APD algorithm. R denotes the APD hyper-parameter, where a larger value of R puts more weight on the diffusion model (and hence more parallel generation) while a smaller value of R puts more weight on AR model (and hence more sequential generation). T/s denotes tokens/step. We use Qwen2.5-0.5B to verify the answers. We compare the accuracy on the GSM8K dataset and the average number of tokens per steps for marginal sampling and joint sampling when combined with APD with different ratios R. Both, Mar+APD and Joint+APD denotes APD restricted to decoding a max of four tokens in each time step. Mar+APD(U), where U stands for unbounded denotes the standard APD algorithm.

B.3 Joint Sampler Loss

	KL@1	KL@2	KL@3
Marginal	0.096	0.164	0.219
Joint	0.061	0.104	0.140

Table 6: In this table we report the KL loss for each speculation head, i.e., given base model features at time t and spec additional ground truth tokens, we compute the KL divergence between the target logits and our predicted (for marginal, the base logits)

B.4 ARCHITECTURE

See Fig 5.

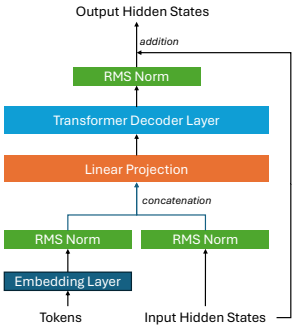

Figure 5: This figure shows the architecture used for our draft model

B.5 GROUNDING PROBABILITIES

We also tried an addition by grounding the predicted probabilities in marginal probabilities. The basic idea can be stated as follows :

Suppose at some denoising step, the current input is $\boldsymbol{x}$ and the predicted marginals are $p(\cdot \mid \boldsymbol{x})$. In the main text we argued that after sampling a token (say at position i), the probability distribution of other positions in the sequence are not yet conditioned on the sampled output at position i. Hence sampling another token from $p(\cdot \mid \boldsymbol{x})$ would lead to an incorrect joint sample. But if the probability of the sampled token at position i is close to 1, i.e., $p^i(x^i \mid \boldsymbol{x}) \approx 1$, then the marginal probabilities of other positions $p(\cdot \mid \boldsymbol{x})$ are already conditioned on this sample. A similar argument was made in Wu et al. (2025) and proved using Boole's inequality.

We can use this fact to ground our predicted probabilities in marginal probabilities. Suppose the probabilities after conditioning on token x^i are p^* and the original probabilities were p_0. Also let the probability of sampling x^i be c, i.e. $p^i(x^i|\boldsymbol{x}) = c$. Then we can say that

$$c \cdot p^* + (1 - c) \cdot \tilde{p} = p_0$$

where p^* is the probability $p(\cdot \mid \boldsymbol{x} \cup x^i)$ and p_0 is the probability $p(\cdot \mid \boldsymbol{x})$, and $\tilde{p}$ is the probability $p(\cdot \mid \boldsymbol{x} \cup \{X^i \neq x^i\})$. Rearranging these terms gives us :

$$p^* = \frac{p_0 - \big((1 - c) \cdot \tilde{p}\big)}{c}$$

Now since $0 \leq \tilde{p} \leq 1$, we get :

$$\frac{p_0 - (1 - c)}{c} \leq p^* \leq \frac{p_0}{c}$$

The same bound holds for our predictions of p^*. This can be implemented as a simple clamping operation. Note that the bound are tighter for $c \approx 1$ with $p^* = p_0$ for $c = 1$.

C ALGORITHMS

Algorithm 3 Marginal Sampling

Input: Prompt $\boldsymbol{x}$, discrete diffusion model f and the language model head $\boldsymbol{W}$, tokens to unmask in each step K, unmasking logic TOP
while $|\mathcal{M}(\boldsymbol{x})| > 0$ **do**
  $\boldsymbol{h} \leftarrow f(\boldsymbol{x})$ ▷ Forward pass through diffusion model
  for $k = 1$ to K **do**
    $i \leftarrow \text{TOP}\{\boldsymbol{W}\boldsymbol{h}^j : j \in \mathcal{M}(\boldsymbol{x})\}$ ▷ Select the position to be unmasked
    $\tilde{x}^i \sim \text{softmax}(\boldsymbol{W}\boldsymbol{h}^i)$ ▷ Sample a token for that position
    $\boldsymbol{x} \leftarrow \boldsymbol{x} \oplus \tilde{x}^i$ ▷ Unmask token by repeating unmasking logic
  end for
end while
return $\boldsymbol{x}$

D ADDITIONAL EXPERIMENTAL DETAILS

D.1 SAMPLING PARAMETERS

For downstream evaluation, we take parameters from the official Dream repository[2]

Dream-7B-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.

[2] https://github.com/DreamLM/Dream