TENSOR-TRAIN JOINT MODELING FOR FEW-STEP DISCRETE DIFFUSION

Anonymous authors
Paper under double-blind review

ABSTRACT

Discrete diffusion promises orders-of-magnitude faster generation than autoregressive (AR) models for sequential discrete data, yet its full potential of few-step generation has remained out of reach due to a fundamental structural limitation. The conditional-independence assumption underlying current discrete diffusion models introduces a systematic *parallelization bias* that compounds with the number of tokens unmasked per step, becoming severe in the few-step regime that fast generation requires. We address this with the first framework for explicit joint distribution modeling in discrete diffusion via *tensor decomposition*, which represents the conditional clean distribution as a low-rank tensor with controllable expressivity. The framework supports both Canonical Polyadic (CPD) and Tensor-Train (TTD) decompositions, and we identify a structural bias of TTD toward dependencies between nearby tokens, formalized through Oseledets' theorem relating TT-rank to unfolding-matrix rank, which is well-suited to sequential data such as natural language and line notations for molecular data. To enable efficient generation, we present an iterative *marginal inference* procedure with specialization for predetermined position schedules. Our framework integrates into pretrained MDMs through lightweight fine-tuning, yielding substantial improvements in few-step generation at a fraction of the cost of training from scratch.

1 INTRODUCTION

Discrete diffusion has emerged as a compelling alternative to AR models for generating sequential discrete data such as text and proteins (Austin et al., 2021; Lou et al., 2024; Sahoo et al., 2024; Shi et al., 2024). Its key promise is parallel token generation, producing multiple tokens per step rather than one. Most notably, Masked Diffusion Models (MDMs) (Sahoo et al., 2024; Shi et al., 2024) have scaled to large language models (Nie et al., 2025; Ye et al., 2025) with competitive quality and faster generation.

In practice, however, this speed potential remains unrealized, as pushing toward few-step generation systematically degrades sample quality. The root cause is structural; to remain tractable, discrete diffusion models represent the conditional clean distribution $p_\theta(\boldsymbol{x}|\boldsymbol{x}_t)$ under the conditional-independence assumption $p_\theta(\boldsymbol{x}|\boldsymbol{x}_t) = \prod_{i=1}^{N} p_\theta(x^i|\boldsymbol{x}_t)$. This assumption introduces a systematic *parallelization bias* (Zhang et al., 2026) when multiple tokens are unmasked simultaneously. The bias compounds with the number of tokens unmasked per step, becoming especially severe in the few-step regime that fast generation requires.

Resolving this bias requires modeling the joint distribution $p_\theta(\boldsymbol{x}|\boldsymbol{x}_t)$ explicitly. However, naïve joint modeling is infeasible since the joint over N tokens with vocabulary size V forms an N-dimensional tensor with V^N entries. Some existing approaches therefore sidestep direct joint modeling, introducing latent factors that induce token correlations (Hayakawa et al., 2025; Xie et al., 2026) or borrowing dependence from auxiliary autoregressive models (Liu et al., 2025; Xu et al., 2025). Compared to AR-auxiliary approaches, our method models the joint distribution explicitly as a tractable parametric family within the discrete diffusion model itself, without requiring any external pretrained model. Also, compared to latent-variable approaches, our explicit parameterization is orthogonal to their implicit one, and the two can be combined for further gains. Li et al. (2026) addresses the same factorization barrier by augmenting the factorized neural output with a tractable probabilistic circuit prior, requiring manual structural design for sequential locality, whereas Tensor-Train decomposition provides the same tractability with a theoretically grounded locality bias.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

In this work, we introduce a framework that explicitly parameterizes the joint distribution $p_\theta(\boldsymbol{x}|\boldsymbol{x}_t)$ via *tensor decomposition* (Kolda & Bader, 2009). Compared to prior work, our framework provides explicit joint modeling within the diffusion model itself, without relying on auxiliary models. Our contributions are four-fold.

- **First explicit joint modeling via tensor decomposition.** To our knowledge, this is the first work to parameterize the conditional clean distribution of discrete diffusion via tensor decomposition, strictly generalizing standard MDMs as the rank-one case.
- **TTD for sequential structure.** We develop the framework for both Canonical Polyadic Decomposition (CPD) (Hitchcock, 1927) and Tensor-Train Decomposition (TTD) (Oseledets, 2011), identifying a structural bias of TTD toward dependencies between nearby tokens, formalized through Oseledets' theorem relating TT-rank to unfolding-matrix rank.
- **Efficient sampling via marginal inference.** Direct sampling from a joint of size V^N is intractable. We present a chain-rule procedure built on iterative *marginal inference* via caching and parallel prefix sums, with specialization for predetermined position schedules.
- **Lightweight fine-tuning of pretrained MDMs.** Our framework integrates into pretrained MDMs through small architectural modifications initialized to preserve the marginal predictions, yielding substantial few-step improvements at a fraction of the cost of training from scratch.

In experiments on text generation (OpenWebText (Gokaslan et al., 2019), LM1B (Chelba et al., 2014)) and molecule generation (QM9 (Blum & Reymond, 2009; Rupp et al., 2012)), our framework delivers substantial few-step improvements across multiple base models, including MDLM (Sahoo et al., 2024) and VADD (Xie et al., 2026). As predicted by our theory, TTD consistently outperforms CPD on these sequential domains, with the largest gains concentrated in the few-step regime. For instance, TTD-based fine-tuning of VADD reduces 8-step generative perplexity on OpenWebText by 32.7% over vanilla VADD.

2 Related Work

Several lines of work attempt to address the parallelization bias arising from the conditional-independence assumption in discrete diffusion. The central challenge is the exponential size of the joint distribution: an N-token sequence over vocabulary V admits V^N possible configurations, making direct modeling intractable.

Auxiliary-model approaches distill dependency information from separately trained autoregressive models via copula-based reweighting (Liu et al., 2025) or energy-based reweighting (Xu et al., 2025). While effective, these methods inherit the inference cost of an external model and become inapplicable in domains lacking strong AR models, such as generation of line notations for molecular data. Our framework, in contrast, requires no additional model.

Latent-variable approaches introduce latent factors that induce token correlations through marginalization, either via a continuous mixing variable (Hayakawa et al., 2025) or a VAE-style latent vector (Xie et al., 2026); our explicit modeling is orthogonal to these implicit approaches and can be combined with them for further gains.

Sampling strategies mitigate the bias without modifying the model, selecting positions via low-discrepancy schedules (Besnier et al., 2025) or adaptively bounding the number of simultaneously unmasked tokens through entropy criteria (Ben-Hamu et al., 2025); these strategies integrate naturally within our framework.

Li et al. (2026) couples the factorized neural output with a tractable probabilistic circuit prior, addressing the same factorization barrier with a different parameterization. Although probabilistic circuits offer general-purpose tractable inference, they require manual structural design for sequential locality, whereas Tensor-Train decomposition provides the same tractability with a theoretically grounded locality bias (Sec. 4.2), making it particularly suited to natural language and line notations for molecular data without further structural engineering.

The parallelization bias has also been studied in the context of speculative decoding for *autoregressive* language models, where Basharin et al. (2025) were the first to apply Canonical Polyadic Decomposition for explicit token-wise dependency modeling. We extend this idea to discrete diffusion for the first time, with two main technical contributions beyond their work: (1) identifying Tensor-

Train Decomposition as particularly suited to sequential structure, with a theoretically grounded inductive bias toward dependencies between nearby tokens; and (2) presenting an efficient sampling procedure that introduces minimal overhead when sampling from the joint distribution.

3 Background & Overview

3.1 Background: Masked Diffusion Models (MDM)

A Masked Diffusion Model (MDM) (Sahoo et al., 2024) is a discrete diffusion model whose forward process is an absorbing process. The absorbing token is commonly referred to as the mask token m and thus lends the model its name. The forward process maps the data distribution $p(\boldsymbol{x})$ to the prior distribution, which is a Dirac delta concentrated at the fully masked sequence $\mathbf{m} = [m, m, \ldots, m]$, as t progresses from 0 to 1. Formally,

$$q(x_t^i|\boldsymbol{x}) = t\,\delta_m(x_t^i) + (1-t)\,\delta_{x^i}(x_t^i), \tag{1}$$

where $x_t^i \in [V] = \{1, 2, \ldots, V\}$ denotes the i-th token at time $t \in [0, 1]$, $\boldsymbol{x} = [x^1, x^2, \ldots, x^N]$ is the clean sequence, $\boldsymbol{x}_t = [x_t^1, \ldots, x_t^N]$ is its partially masked version, and δ_a denotes the point mass at a.

The model is trained to predict the original values of the masked tokens in a partially masked sequence, typically under an assumption of conditional independence across tokens. Namely,

$$p_\theta(\boldsymbol{x}|\boldsymbol{x}_t) = \prod_{i=1}^{N} p_\theta(x^i|\boldsymbol{x}_t). \tag{2}$$

Given a partially masked sequence $\boldsymbol{x}_t$, for every position i and every $x^i \in [V]$, the model predicts the marginal probability $p_\theta(x^i|\boldsymbol{x}_t)$ that the i-th token of the clean sequence equals x^i. Under the factorization, the negative log-likelihood (NLL) loss

$$\mathcal{L}_{\text{NLL}} = \mathbb{E}_{t\sim U(0,1),\ \boldsymbol{x}\sim p(\boldsymbol{x}),\ \boldsymbol{x}_t\sim q(\boldsymbol{x}_t|\boldsymbol{x})}[-\log p_\theta(\boldsymbol{x}|\boldsymbol{x}_t)] \tag{3}$$

reduces to the sum of per-position marginal NLLs:

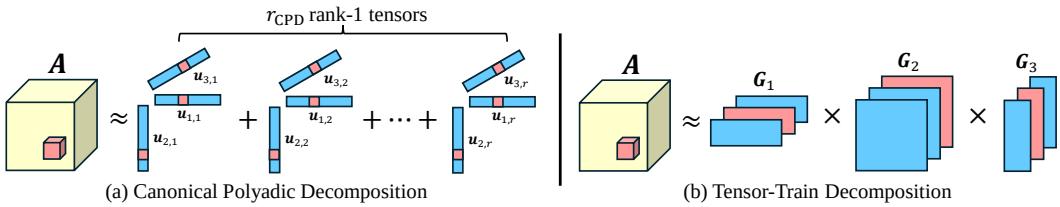
$$\mathcal{L}_{\text{marginal}} = \mathbb{E}\left[\sum_{i=1}^{N} -\log p_\theta(x^i|\boldsymbol{x}_t)\right]. \tag{4}$$

Generation proceeds by initializing a fully masked sequence and iteratively unmasking tokens until a clean, mask-free sequence is obtained.

3.2 Method Overview

Despite the promising progress of masked diffusion models, a key limitation lies in the conditional-independence assumption (Eq. (2)) imposed on $p_\theta(\boldsymbol{x}|\boldsymbol{x}_t)$, which introduces a systematic *parallelization bias* (Zhang et al., 2026) that becomes especially severe in the few-step regime. Explicitly modeling the joint distribution $p_\theta(\boldsymbol{x}|\boldsymbol{x}_t)$, however, is fundamentally a problem of scale. In continuous diffusion, the joint admits a tractable parameterization as a multivariate Gaussian. In the discrete setting, by contrast, the joint corresponds to an N-dimensional tensor of size V^N, which rapidly becomes intractable as the vocabulary size V and the sequence length N grow.

To address this challenge, we propose to model $p_\theta(\boldsymbol{x}|\boldsymbol{x}_t)$ explicitly via *tensor decomposition*. In Sec. 4, we investigate two decompositions particularly suited to this purpose—Canonical Polyadic Decomposition (CPD) (Hitchcock, 1927) and Tensor-Train Decomposition (TTD) (Oseledets, 2011)—both supporting efficient training and sampling with only $O(VN)$ memory. We describe how each parameterizes the joint distribution and how generation is carried out under it. Our analysis identifies a structural limitation of CPD that motivates the use of TTD, whose form is naturally aligned with dependencies between *nearby* tokens, a property especially well suited to sequential data such as natural language (Futrell et al., 2015; Liu et al., 2017) and line notations for molecular data. Note that either decomposition reduces to the standard marginal probability formulation (Eq. (2)) in the rank-one special case. To our knowledge, this is the first work introducing tensor decomposition to joint probability modeling in discrete diffusion.



(a) Canonical Polyadic Decomposition (b) Tensor-Train Decomposition

Figure 1: Comparison of tensor decomposition methods: Canonical Polyadic Decomposition (CPD) and Tensor Train Decomposition (TTD).

4 Joint Probability Modeling via Tensor Decomposition

In this section, we describe how the joint distribution $p_\theta(\boldsymbol{x}|\boldsymbol{x}_t)$ can be modeled using two tensor decompositions: Canonical Polyadic Decomposition (CPD; Sec. 4.1) and Tensor-Train Decomposition (TTD; Sec. 4.2). We discuss the advantages of TTD over CPD in capturing dependencies between nearby tokens, a property crucial for many practical domains such as text and line notations for molecular data. Building on the formulations, Sec. 4.3 describes the sampling procedure under either decomposition.

4.1 Modeling via Canonical Polyadic Decomposition (CPD)

Given an order-N tensor $\boldsymbol{A}$, Canonical Polyadic Decomposition (CPD) (Hitchcock, 1927) approximates $\boldsymbol{A}$ by a sum of r rank-1 tensors, each formed as the outer product of N vectors:

$$\boldsymbol{A}(\nu_1, \ldots, \nu_N) \approx \sum_{\alpha=1}^{r} \prod_{i=1}^{N} \boldsymbol{u}_{i,\alpha}(\nu_i), \tag{5}$$

where each $\boldsymbol{u}_{i,\alpha} \in \mathbb{R}^V$ is a one-dimensional factor vector, and r is referred to as the *CP-rank*, which we may write as r_{CPD}. Building on this, we model the joint distribution $p_\theta(\boldsymbol{x}|\boldsymbol{x}_t)$ over a sequence of N tokens via CPD, with an additional rank weight (as in the autoregressive setting of Basharin et al. (2025)):

$$p_\theta(\boldsymbol{x}|\boldsymbol{x}_t) = \sum_{\alpha=1}^{r} w_\theta^{(\alpha)}(\boldsymbol{x}_t) \prod_{i=1}^{N} p_\theta^{(\alpha,i)}(x^i|\boldsymbol{x}_t), \tag{6}$$

where $w_\theta^{(\alpha)}(\boldsymbol{x}_t) \geq 0$ is the weight produced for the α-th rank-1 component, and $p_\theta^{(\alpha,i)}(x^i|\boldsymbol{x}_t) \geq 0$ is the corresponding output for token x^i at position i in that component. For normalization, we enforce

$$\sum_{\alpha=1}^{r} w_\theta^{(\alpha)}(\boldsymbol{x}_t) = 1, \qquad \sum_{x^i=1}^{V} p_\theta^{(\alpha,i)}(x^i|\boldsymbol{x}_t) = 1, \tag{7}$$

which together ensure that $p_\theta(\boldsymbol{x}|\boldsymbol{x}_t)$ is a valid probability distribution while maintaining expressibility (proof in Appendix A). These constraints further allow $p_\theta^{(\alpha,i)}(x^i|\boldsymbol{x}_t)$ to be interpreted as the marginal probability of x^i at position i within the α-th component.

Note that when $r = 1$, the formulation reduces to the standard MDM, which predicts only per-position marginal probabilities. Moreover, any lower-rank representation with $r' < r$ can be recovered by zeroing the surplus weights, so the family of joint distributions expressible at rank r strictly contains those expressible at any lower rank.

In practice, r must be kept small for computational and memory efficiency. To maximize the benefit of a low-rank approximation, the key is to model inter-token dependencies in a manner aligned with the structure of the data modality. For natural language, it is well known that *nearby* tokens are the most strongly correlated (Futrell et al., 2015; Liu et al., 2017). To capture such local dependencies effectively, we next introduce Tensor-Train Decomposition (TTD) and discuss its advantages over CPD.

4.2 Modeling via Tensor-Train Decomposition (TTD)

Tensor-Train Decomposition (TTD) (Oseledets, 2011) approximates an order-N tensor $\boldsymbol{A}$ by a sequence of *cores* $\boldsymbol{G}_1, \ldots, \boldsymbol{G}_N$, with each entry obtained as a product of the corresponding matrix

slices:

$$\boldsymbol{A}(\nu_1, \ldots, \nu_N) \approx \boldsymbol{G}_1(\nu_1)\, \boldsymbol{G}_2(\nu_2) \cdots \boldsymbol{G}_N(\nu_N), \tag{8}$$

where each core $\boldsymbol{G}_i \in \mathbb{R}^{V \times r_{i-1} \times r_i}$ has matrix slices $\boldsymbol{G}_i(\nu_i) := \boldsymbol{G}_i(\nu_i, :, :) \in \mathbb{R}^{r_{i-1} \times r_i}$, with boundary ranks $r_0 = r_N = 1$ so that the right-hand side is a scalar. The tuple of internal ranks $(r_1, r_2, \ldots, r_{N-1})$ is referred to as the ***TT-rank*** of the decomposition.

We adopt this form to model the joint distribution $p_\theta(\boldsymbol{x}|\boldsymbol{x}_t)$, identifying each tensor mode with a sequence position. Each core $\boldsymbol{G}_{i,\theta}(x^i|\boldsymbol{x}_t) \in \mathbb{R}_{\geq 0}^{r_{i-1} \times r_i}$ is conditioned on the masked input. For practical implementation, we let every core share the common shape $V \times r \times r$, i.e., $r_0 = r_1 = \cdots = r_N = r$ (which we may also write as r_{TTD}), and contract the boundary dimensions with all-ones vectors $\mathbf{1}_r \in \mathbb{R}^r$ to recover a scalar:

$$p_\theta(\boldsymbol{x}|\boldsymbol{x}_t) = \tfrac{1}{r} \mathbf{1}_r^\top\, \boldsymbol{G}_{1,\theta}(x^1|\boldsymbol{x}_t)\, \boldsymbol{G}_{2,\theta}(x^2|\boldsymbol{x}_t) \cdots \boldsymbol{G}_{N,\theta}(x^N|\boldsymbol{x}_t)\, \mathbf{1}_r. \tag{9}$$

To ensure that $p_\theta(\boldsymbol{x}|\boldsymbol{x}_t)$ is a valid probability distribution, we impose the following normalization constraint on each core, which also maintains expressibility (proof in Appendix A):

$$\sum_{x^i, k} \boldsymbol{G}_{i,\theta}(x^i, j, k|\boldsymbol{x}_t) = 1, \quad \forall i, j. \tag{10}$$

As with CPD, when $r = 1$, the formulation reduces to the standard MDM, which predicts only per-position marginal probabilities. Moreover, any lower-rank representation with $r' < r$ can be recovered by zeroing surplus columns of the cores, so the family of joint distributions expressible at rank r strictly contains those expressible at any lower rank.

A key advantage of TTD over CPD is its structural bias toward modeling dependencies between nearby tokens. To make this precise, we recall a result from Oseledets (2011) that connects each TT-rank r_i to the dependency structure of $\boldsymbol{A}$.

For each split position $i \in \{1, \ldots, N-1\}$, partition the N token positions into two contiguous blocks: a *left block* consisting of positions $1, \ldots, i$, and a *right block* consisting of positions $i+1, \ldots, N$. The *unfolding matrix* $\boldsymbol{A}_i$ at this split is obtained by reshaping $\boldsymbol{A}$ into a matrix whose rows are indexed by the joint values of the left block and whose columns by those of the right:

$$\boldsymbol{A}_i \in \mathbb{R}^{V^i \times V^{N-i}}, \qquad \boldsymbol{A}_i\big([\nu_1, \ldots, \nu_i], [\nu_{i+1}, \ldots, \nu_N]\big) = \boldsymbol{A}(\nu_1, \ldots, \nu_N). \tag{11}$$

The rank of $\boldsymbol{A}_i$ captures how much information must flow across the cut at position i: a small $\operatorname{rank}(\boldsymbol{A}_i)$ means that the joint values of the left block can be summarized into a low-dimensional vector that fully determines the distribution over the right block, indicating that the dependence between the two sides is limited.

Oseledets (2011, Theorem 2.1) establishes that the minimum TT-rank required at level i is exactly the rank of the corresponding unfolding matrix:

$$r_i = \operatorname{rank}(\boldsymbol{A}_i). \tag{12}$$

Intuitively, this is because any TT representation of $\boldsymbol{A}$ already factorizes $\boldsymbol{A}_i$ into a product of a $V^i \times r_i$ matrix (built from the first i cores) and a $r_i \times V^{N-i}$ matrix (built from the remaining cores), so $r_i \geq \operatorname{rank}(\boldsymbol{A}_i)$; the reverse direction is shown constructively in the same theorem.

The crucial implication is that each r_i depends only on the dependence structure across the i-th cut, not on the overall complexity of $\boldsymbol{A}$. For sequential data dominated by *local* dependencies, the unfolding ranks $\operatorname{rank}(\boldsymbol{A}_i)$ can remain small at every cut and not scale with the sequence length, so a small TT-rank r can suffice for an accurate approximation. We also provide quantitative evidence linking TTD performance with local dependencies in Sec. E.1.

CPD, by contrast, treats all positions symmetrically and offers no analogous reduction when locality is the dominant structure of the data. As a concrete illustration of how extreme this asymmetry can be, Appendix B constructs a specific distribution over $2N$ tokens for which $r_{\text{CPD}} = 2^N$ but $r_{\text{TTD}} = 2$.

4.3 Efficient Sampling via Iterative Marginal Inference

The tensor-decomposed forms (CPD and TTD) of $p_\theta(\boldsymbol{x}|\boldsymbol{x}_t)$ admit efficient evaluation at any $\boldsymbol{x}$, so training proceeds identically to a standard MDM with the same NLL loss (Eq. (3)). Sampling, however, is nontrivial: naively sampling from $p_\theta(\boldsymbol{x}|\boldsymbol{x}_t)$ would require evaluating it at all V^N possible

sequences, an exponential cost in the sequence length N. For efficient sampling, we present a general procedure based on the chain rule, followed by a special case that exploits additional structure.

The chain rule of probability decomposes the joint into a sequence of conditionals. For any ordering $(i_1, \ldots, i_K)$ of the K positions,

$$p(x^{i_1}, \ldots, x^{i_K} | \boldsymbol{x}_t) = \prod_{j=1}^{K} p(x^{i_j} | x^{i_1}, \ldots, x^{i_{j-1}}, \boldsymbol{x}_t). \tag{13}$$

We can therefore sample the subsequence one position at a time by conditioning on previously sampled positions and repeatedly applying *marginal inference*. Specifically, given a set $S \subseteq \{1, \ldots, N\}$ of conditioned positions with values $\boldsymbol{x}_S := \{x^j\}_{j \in S}$, we compute the marginal $p(x^i | \boldsymbol{x}_S, \boldsymbol{x}_t)$ at every unconditioned position $i \notin S$. Sampling the full sequence then proceeds by initializing $S = \emptyset$ and iteratively (i) performing marginal inference, (ii) selecting a position $i \notin S$, (iii) sampling $x^i \sim p(x^i | \boldsymbol{x}_S, \boldsymbol{x}_t)$, and (iv) adding i to S. The position selection in step (ii) can follow any logit-based strategy (e.g., top probability-based (Zheng et al., 2024) or entropy-based (Ye et al., 2025) selection). The number of tokens sampled in a single step can be either fixed or determined by an adaptive sampler such as EB-Sampler (Ben-Hamu et al., 2025). We now describe how to perform marginal inference under each decomposition.

CPD. For each position j and rank component α, define the *conditioning factor*

$$\bar{p}_\theta^{(\alpha,j)} = \begin{cases} p_\theta^{(\alpha,j)}(x^j | \boldsymbol{x}_t) & \text{if } j \in S, \\ 1 & \text{otherwise.} \end{cases} \tag{14}$$

Substituting into the CPD form (Eq. (6)), for any $i \notin S$,

$$p(x^i | \boldsymbol{x}_S, \boldsymbol{x}_t) = \frac{p(x^i, \boldsymbol{x}_S | \boldsymbol{x}_t)}{p(\boldsymbol{x}_S | \boldsymbol{x}_t)} \propto p(x^i, \boldsymbol{x}_S | \boldsymbol{x}_t) = \sum_{\alpha=1}^{r} w_\theta^{(\alpha)}(\boldsymbol{x}_t)\, p_\theta^{(\alpha,i)}(x^i | \boldsymbol{x}_t) \prod_{j=1}^{N} \bar{p}_\theta^{(\alpha,j)}. \tag{15}$$

Since $\bar{p}_\theta^{(\alpha,i)} = 1$ for $i \notin S$, the product $\prod_{j=1}^{N} \bar{p}_\theta^{(\alpha,j)}$ does not depend on i and can therefore be precomputed once and reused across all positions, enabling parallel evaluation of $p(x^i, \boldsymbol{x}_S | \boldsymbol{x}_t)$ for every $i \notin S$. The marginal $p(x^i | \boldsymbol{x}_S, \boldsymbol{x}_t)$ is then obtained by normalizing $p(x^i, \boldsymbol{x}_S | \boldsymbol{x}_t)$ over the vocabulary. (Division by $p(\boldsymbol{x}_S | \boldsymbol{x}_t)$ is also possible but unnecessary and less numerically stable.)

TTD. Analogously, for each position j, define the *contracted core*

$$\bar{\boldsymbol{G}}_j = \begin{cases} \boldsymbol{G}_{j,\theta}(x^j | \boldsymbol{x}_t) & \text{if } j \in S, \\ \sum_{x^j=1}^{V} \boldsymbol{G}_{j,\theta}(x^j | \boldsymbol{x}_t) & \text{otherwise.} \end{cases} \tag{16}$$

When $j \notin S$, $\bar{\boldsymbol{G}}_j$ does not depend on the chain-rule conditioning $\boldsymbol{x}_S$ and can be cached across queries. For any $i \notin S$,

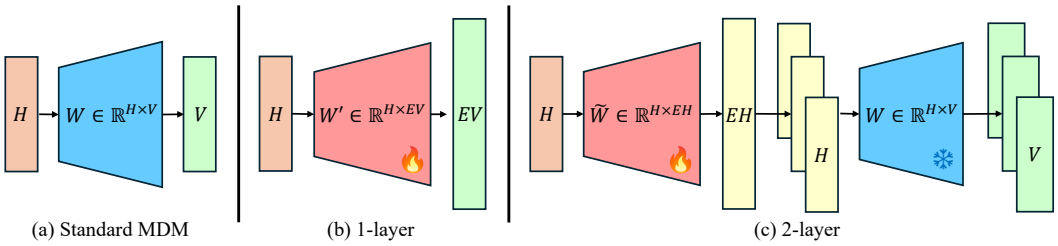
$$p(x^i, \boldsymbol{x}_S | \boldsymbol{x}_t) = \tfrac{1}{r}\, \mathbf{1}_r^\top\, \bar{\boldsymbol{G}}_1 \cdots \bar{\boldsymbol{G}}_{i-1}\, \boldsymbol{G}_{i,\theta}(x^i | \boldsymbol{x}_t)\, \bar{\boldsymbol{G}}_{i+1} \cdots \bar{\boldsymbol{G}}_N\, \mathbf{1}_r. \tag{17}$$

The left-cumulative products $\bar{\boldsymbol{G}}_1 \cdots \bar{\boldsymbol{G}}_{i-1}$ and right-cumulative products $\bar{\boldsymbol{G}}_{i+1} \cdots \bar{\boldsymbol{G}}_N$ for all i can be obtained via parallel prefix sums. Normalizing $p(x^i, \boldsymbol{x}_S | \boldsymbol{x}_t)$ over the vocabulary then yields $p(x^i | \boldsymbol{x}_S, \boldsymbol{x}_t)$, again avoiding the explicit computation of $p(\boldsymbol{x}_S | \boldsymbol{x}_t)$.

4.3.1 Special Case: Predetermined Sampling Positions

While the method above samples one position at a time, in many settings—random ordering, fixed orderings such as left-to-right — the positions to be sampled in the current step are determined before any computation, allowing them to be sampled *jointly*. Let $T \subseteq \{1, \ldots, N\}$ denote the masked positions to be sampled jointly in the same step, with values $\boldsymbol{x}_T := \{x^j\}_{j \in T}$. For CPD:

$$p(\boldsymbol{x}_T | \boldsymbol{x}_t) = \sum_{\alpha=1}^{r} w_\theta^{(\alpha)}(\boldsymbol{x}_t) \prod_{j \in T} p_\theta^{(\alpha,j)}(x^j | \boldsymbol{x}_t) \tag{18}$$



(a) Standard MDM (b) 1-layer (c) 2-layer

Figure 2: Architectural modifications for fine-tuning MDMs.

Sampling therefore requires evaluating $p_\theta^{(\alpha,j)}(x^j|\boldsymbol{x}_t)$ only at positions $j \in T$.

With TTD, positions $j \notin T$ enter $p(\boldsymbol{x}_T|\boldsymbol{x}_t)$ only through the contracted cores $\bar{\boldsymbol{G}}_j$. By absorbing each $\bar{\boldsymbol{G}}_j$ into a neighboring core, the remaining cores form a TT representation over only the $|T|$ positions in T. On this we can run a highly optimized version of the general TTD method, by caching cumulative products and computing $p(x^i|\boldsymbol{x}_S, \boldsymbol{x}_t)$ for only the next immediate value of i.

The reduction above assumes access to $\bar{\boldsymbol{G}}_j$ for each $j \notin T$. Computing it from the definition by outputting $\boldsymbol{G}_{j,\theta}(x^j|\boldsymbol{x}_t)$ for all V values of x^j and summing is computationally wasteful. We sidestep this overhead by introducing an additional head $\bar{\boldsymbol{G}}_{j,\theta}(\boldsymbol{x}_t)$ that directly produces the contracted core, trained to be consistent with $\boldsymbol{G}_{j,\theta}(x^j|\boldsymbol{x}_t)$:

$$\mathcal{L}_{\mathrm{C}} = \frac{1}{N}\sum_{j=1}^{N}\left\|\bar{\boldsymbol{G}}_{j,\theta}(\boldsymbol{x}_t) - \mathrm{stopgrad}\Big(\sum_{x^j}\boldsymbol{G}_{j,\theta}(x^j|\boldsymbol{x}_t)\Big)\right\|_F^2 \tag{19}$$

then optimize

$$\mathcal{L} = \mathcal{L}_{\mathrm{NLL}} + \mathcal{L}_{\mathrm{C}}. \tag{20}$$

4.4 Extension to Latent-Variable Approaches

Our framework is flexible: beyond a base MDM, it can also be combined with latent-variable approaches. We illustrate this with Variational Autoencoding Discrete Diffusion (VADD) (Xie et al., 2026), a few-step MDM framework based on a latent variable model. Integrating our explicit modeling of $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{x}_t)$ into VADD yields the training loss

$$\widehat{\mathcal{L}}_\lambda(\boldsymbol{x}_0;\boldsymbol{\theta},\boldsymbol{\phi}) = \int_0^1 \mathbb{E}_{q(\boldsymbol{x}_t|\boldsymbol{x}_0)}\mathbb{E}_{r_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}_0,\boldsymbol{x}_t)}\frac{-\alpha_t'}{1-\alpha_t}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_0|\boldsymbol{x}_t,\boldsymbol{z}) - \lambda\log\left(\frac{r_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}_0,\boldsymbol{x}_t)}{p(\boldsymbol{z})}\right)\right]\mathrm{d}t, \tag{21}$$

with the sampling procedure

$$\boldsymbol{z} \sim p(\boldsymbol{z}), \qquad \boldsymbol{x}_{t_{i-1}} \sim p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_{i-1}}|\boldsymbol{x}_{t_i},\boldsymbol{z}),$$

where our tensor decomposition parameterizes both $p_{\boldsymbol{\theta}}(\boldsymbol{x}_0|\boldsymbol{x}_t,\boldsymbol{z})$ and $p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_{i-1}}|\boldsymbol{x}_{t_i},\boldsymbol{z})$. Results for this combination are reported in Sec. 5.1.

4.5 Fine-Tuning Pretrained MDMs and Implementation Details

Our tensor-decomposition framework can be integrated into pretrained MDMs via fine-tuning. We describe the necessary architectural modifications and other implementation details below.

Architecture. A standard MDM produces NV logits, but tensor-decomposed forms require EVN, where $E = r$ (CPD; plus r rank weights) or $E = r^2$ (TTD). Our *1-layer* method replaces the final head $\boldsymbol{W} \in \mathbb{R}^{H\times V}$ with $\boldsymbol{W}' \in \mathbb{R}^{H\times EV}$, which is costly for large V. Following Basharin et al. (2025), the *2-layer* method instead inserts $\widetilde{\boldsymbol{W}} \in \mathbb{R}^{H\times EH}$ before the final head, then applies $\boldsymbol{W}$ to each of the E outputs, which significantly reduces the parameter count since $H \ll V$ in language models. Schematics of different architectures are shown in Fig. 2.

Initialization. We initialize the added weights to preserve pretrained marginal predictions: $\boldsymbol{W}'$ as E copies of $\boldsymbol{W}$ (1-layer), or $\widetilde{\boldsymbol{W}}$ as E stacked identity matrices with $\boldsymbol{W}$ frozen (2-layer). Small Gaussian noise is added to break rank-symmetry.

Table 1: Generative perplexity (GPT-2) on OpenWebText (Gokaslan et al., 2019).

Method	Timesteps	Generative Perplexity (↓)	Entropy
MDLM-based			
MDLM	8 / 16 / 32 / 64 / 128	840.57 / 347.10 / 195.09 / 146.09 / 122.08	7.77 / 7.67 / 7.60 / 7.58 / 7.56
EDLM	8 / 16 / 32 / 64 / 128	891.18 / 367.48 / 212.02 / 153.79 / 129.38	7.83 / 7.73 / 7.66 / 7.62 / 7.60
Di4C	8 / 16 / 32 / 64 / 128	832.52 / 360.89 / 207.50 / 152.63 / 127.31	7.79 / 7.70 / 7.63 / 7.60 / 7.57
MDLM+CPD (Ours)	8 / 16 / 32 / 64 / 128	848.87 / 358.25 / 205.46 / 148.10 / 126.85	7.81 / 7.71 / 7.65 / 7.60 / 7.60
MDLM+TTD (Ours)	8 / 16 / 32 / 64 / 128	**636.98 / 295.43 / 183.43 / 140.97 / 119.20**	7.75 / 7.66 / 7.60 / 7.57 / 7.54
SEDD-based			
DCD	8 / 16 / 32 / 64 / 128	252.11 / 240.75 / 201.75 / 189.09 / 176.74	8.65 / 8.76 / 8.83 / 8.89 / 8.89
VADD-based			
VADD	8 / 16 / 32 / 64 / 128	331.69 / 204.31 / 151.84 / 132.09 / 123.09	7.05 / 7.02 / 6.98 / 6.98 / 6.97
VADD+CPD (Ours)	8 / 16 / 32 / 64 / 128	299.17 / 180.72 / 135.97 / 118.11 / 108.57	6.93 / 6.90 / 6.86 / 6.86 / 6.84
VADD+TTD (Ours)	8 / 16 / 32 / 64 / 128	**223.19 / 153.85 / 122.34 / 109.80 / 101.70**	6.82 / 6.80 / 6.78 / 6.78 / 6.76

Table 2: Generative perplexity (GPT-2) on LM1B (Chelba et al., 2014).

Method	Timesteps	Generative Perplexity (↓)	Entropy
VADD	1 / 2 / 4 / 8 / 16	1096.39 / 664.16 / 414.02 / 299.39 / 251.03	6.94 / 6.90 / 6.90 / 6.93 / 6.93
VADD+CPD (Ours)	1 / 2 / 4 / 8 / 16	1099.93 / 659.87 / 419.69 / 298.66 / 249.77	6.95 / 6.90 / 6.90 / 6.93 / 6.93
VADD+TTD (Ours)	1 / 2 / 4 / 8 / 16	**850.64 / 532.43 / 369.56 / 284.86 / 247.40**	6.99 / 6.93 / 6.94 / 6.95 / 6.96

Table 3: Rank size ablation on OpenWebText (Gokaslan et al., 2019), measured by generative perplexity (GPT-2).

Method	Timesteps	Generative Perplexity (↓)	Entropy
MDLM	8 / 16 / 32 / 64 / 128	840.57 / 347.10 / 195.09 / 146.09 / 122.08	7.77 / 7.67 / 7.60 / 7.58 / 7.56
MDLM+TTD (r=2)	8 / 16 / 32 / 64 / 128	703.02 / 304.57 / 185.11 / **140.47** / 120.74	7.75 / 7.63 / 7.59 / 7.55 / 7.55
MDLM+TTD (r=3)	8 / 16 / 32 / 64 / 128	664.63 / 299.72 / 187.73 / 140.82 / 123.08	7.78 / 7.66 / 7.61 / 7.58 / 7.58
MDLM+TTD (r=4)	8 / 16 / 32 / 64 / 128	**636.98** / **295.43** / **183.43** / 140.97 / **119.20**	7.75 / 7.66 / 7.60 / 7.57 / 7.54

Masked-token-only modeling. Let $\mathcal{M} := \{i : x_t^i = m\}$ denote the set of masked token indices. During the forward process, unmasked tokens remain unchanged (*i.e.,* $x_t^i = x^i$ for $i \notin \mathcal{M}$), so the likelihood factorizes as $p_\theta(\boldsymbol{x}|\boldsymbol{x}_t) = p_\theta(\boldsymbol{x}_\mathcal{M}|\boldsymbol{x}_t) \prod_{i \notin \mathcal{M}} \delta_{x_t^i}(x^i)$. We therefore restrict network forward passes to predicting $p_\theta(\boldsymbol{x}_\mathcal{M}|\boldsymbol{x}_t)$ over masked tokens only for efficiency. The NLL loss is normalized by $|\mathcal{M}|$ accordingly.

5 Experimental Results

5.1 Text Generation

We demonstrate the application of our methods on both MDLM (Sahoo et al., 2024) and VADD (Xie et al., 2026). For MDLM (Sahoo et al., 2024), we use the official checkpoint trained on OpenWebText (OWT) (Gokaslan et al., 2019) as the base model, and for VADD (Xie et al., 2026) we train a base model from scratch for both OWT and LM1B (Chelba et al., 2014). These models are augmented with the 2-layer architecture (Basharin et al., 2025) before fine-tuning, with $r_{\text{CPD}} = 16, r_{\text{TTD}} = 4$. For training and sampling with VADD and other baselines (Xu et al., 2025; Hayakawa et al., 2025), we use their official implementation, while fixing the numerical issue in categorical sampling reported by Zheng et al. (2025), using 64-bit precision for logits. Additional details on training and sampling can be found in App. D.1.

We sample 1024 samples from each using random ordering, then measure the generative perplexity with gpt2-large (Radford et al., 2019), and the entropy over every token in the 1024 generated sequences as a measure of diversity. The sample length is 1024 for OWT (Gokaslan et al., 2019) and 128 for LM1B (Chelba et al., 2014).

Results on OWT (Gokaslan et al., 2019) show that TTD significantly improves over base MDLM (Sahoo et al., 2024), reducing generative perplexity by 24.22% and 14.88% in the challenging 8- and 16-step regimes, while CPD yields modest to no improvement (Tab. 1, Top). A similar trend holds for VADD+TTD on the same dataset (Tab. 1, Bottom) and on LM1B (Chelba et al., 2014), where large perplexity gaps persist across all sampling steps (Tab. 2). We conjecture

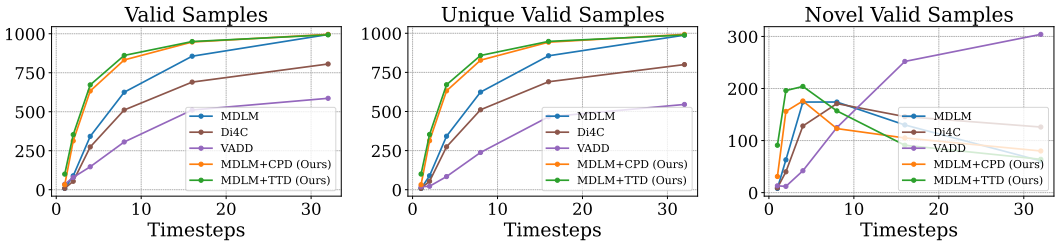


Figure 3: Molecule generation on QM9 (Blum & Reymond, 2009; Rupp et al., 2012). The proposed method consistently achieves higher validity than baselines, including few-step regimes.

that the gain of TTD on VADD stems from two directions of joint probability modeling: VADD captures high-level semantics via a latent variable, while TTD models low-level dependencies between neighboring tokens through lightweight fine-tuning with the objective in Sec. 4.3.1. We also provide further quantitative evidence linking TTD performance with local dependencies in Sec. E.1.

We further analyze the proposed method along two axes: rank size impact and runtime overhead. We begin by ablating the rank size using MDLM (Sahoo et al., 2024) trained on OWT (Gokaslan et al., 2019), by varying the rank size from $r = 2$ to $r = 4$. The results are summarized in Tab. 3. As shown, generative perplexity largely improves as rank increases, validating that greater expressibility of the decomposition helps in capturing the joint distribution.

Due to the large V in language models, naïvely applying our methods can impose a significant overhead. However, the techniques discussed in Sec. 4.3.1 substantially reduce this cost. To demonstrate this, we measure sampling time across methods at 128 timesteps, and report the results in Tab. 4. Our methods are faster than other baselines, with TTD incurring only 23% overhead and CPD's overhead being close to zero. Moreover, we stress that the overhead stems from the size of output tensors rather than model size, and thus does not scale with model depth; we expect it to be minimal for diffusion language models of significantly larger sizes.

Table 4: Sampling time on OpenWebText (Gokaslan et al., 2019), measured with 128 timesteps.

Method	Sec/Seq (↓)
MDLM	2.39
EDLM	6.71
Di4C	3.63
MDLM+CPD (Ours)	2.37
MDLM+TTD (Ours)	2.96

5.2 Molecule Generation

We evaluate on the molecule generation task using QM9 (Blum & Reymond, 2009; Rupp et al., 2012), with MDLM (Sahoo et al., 2024), parameterized using DiT (Peebles & Xie, 2023), as the base model. Given the smaller vocabulary size ($V = 40$) relative to text generation, a 1-layer architecture suffices, with $r_{\text{CPD}} = 64, r_{\text{TTD}} = 8$. Training and sampling details are described in App. D.2.

For the DiT architecture, we modify the rotary positional encoding to apply the rotary matrices to values as well. We stress that this is crucial for any MDM-like sampling methods. Without such a modification, the first sampling step collapses entirely for both the baseline MDLM and our methods due to the uniformity of the inputs in every position. The necessity of such a modification is proved in App. C.

We evaluate each method by sampling 1024 molecules with random ordering across varying numbers of sampling steps, then compute validity, and uniqueness and novelty among valid samples. These metrics are summarized in Fig. 3. We find that our methods bring significant improvements despite small increases in model size. Additional experiments, including sampling strategy comparisons and rank size ablations, are provided in App. E.2. We observe that, as predicted from our discussion in Sec. 4, TTD outperforms CPD of the same output size, and that this gap is the largest in left-to-right generation, where the structural bias of TTD can be exploited to the fullest extent.

6 Conclusion

We presented the first framework for explicit joint distribution modeling in discrete diffusion, parameterizing the conditional clean distribution $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{x}_t)$ as a low-rank tensor via CPD or TTD and thus generalizing standard MDMs. Our analysis identifies a structural bias of TTD toward dependencies between nearby tokens, making it particularly well-suited to sequential data. Through an efficient sampling procedure and a lightweight fine-tuning recipe, the framework integrates with any base MDM or latent-variable method and substantially improves few-step generation on both text and molecule benchmarks.

AI USE STATEMENT

In this work, we used generative AI tools for providing critical ingredients for proving mathematical claims and implementing methods. We have not used generative AI tools for help developing theoretical models or conceptual frameworks, formulating mathematical claims, assistance in the writing of proofs, proposing or refining hypotheses, designing or providing feedback on research methodology or experiments, assistance with translation, cleaning and reformatting dataset, supporting qualitative and thematic data analysis or interpreting results, and generating synthetic data sets are not applicable to this work. Additionally, we used generative AI tools for optimizing code with well-defined requirements, surveying previous literature and finding minor mistakes in our writing. We have reviewed all AI-assisted work through manual checking and code testing. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. In *NeurIPS*, 2021.

Artem Basharin, Andrei Chertkov, and Ivan Oseledets. Faster language models with better multi-token prediction using tensor decomposition. *arXiv preprint arXiv:2410.17765*, 2025.

Heli Ben-Hamu, Itai Gat, Daniel Severo, Niklas S Nolte, and Brian Karrer. Accelerated sampling from masked diffusion models via entropy bounded unmasking. In *NeurIPS*, 2025.

Victor Besnier, Mickael Chen, David Hurych, Eduardo Valle, and Matthieu Cord. Halton scheduler for masked generative image transformer. In *ICLR*, 2025.

L. C. Blum and J.-L. Reymond. 970 million druglike small molecules for virtual screening in the chemical universe database GDB-13. *J. Am. Chem. Soc.*, 131:8732, 2009.

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005*, 2014.

Richard Futrell, Kyle Mahowald, and Edward Gibson. Large-scale evidence of dependency length minimization in 37 languages. *Proceedings of the National Academy of Sciences*, 112(33):10336–10341, 2015.

Aaron Gokaslan, Vanya Cohen, Ellie Pavlick, and Stefanie Tellex. Openwebtext corpus. `http://Skylion007.github.io/OpenWebTextCorpus`, 2019.

Satoshi Hayakawa, Yuhta Takida, Masaaki Imaizumi, Hiromi Wakaki, and Yuki Mitsufuji. Distillation of discrete diffusion through dimensional correlations. In *ICML*, 2025.

Frank L. Hitchcock. The expression of a tensor or a polyadic as a sum of products. *Journal of Mathematics and Physics*, 6(1-4):164–189, 1927.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

Tamara G. Kolda and Brett W. Bader. Tensor decompositions and applications. *SIAM Review*, 51 (3):455–500, 2009.

Ian Li, Zilei Shao, Benjie Wang, Rose Yu, Guy Van den Broeck, and Anji Liu. Breaking the factorization barrier in diffusion language models. *arXiv preprint arXiv:2603.00045*, 2026.

Anji Liu, Oliver Broadrick, Mathias Niepert, and Guy Van den Broeck. Discrete copula diffusion. In *ICLR*, 2025.

Haitao Liu, Chunshan Xu, and Junying Liang. Dependency distance: A new perspective on syntactic patterns in natural languages. *Physics of Life Reviews*, 21:171–193, 2017.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. In *ICML*, 2024.

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. In *NeurIPS*, 2025.

Ivan V Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *ICCV*, 2023.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

M. Rupp, A. Tkatchenko, K.-R. Müller, and O. A. von Lilienfeld. Fast and accurate modeling of molecular atomization energies with machine learning. *Physical Review Letters*, 108:058301, 2012.

Subham S Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. In *NeurIPS*, 2024.

Jiaxin Shi, Kehang Han, Zhe Wang, Arnaud Doucet, and Michalis Titsias. Simplified and generalized masked diffusion for discrete data. In *NeurIPS*, 2024.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Tianyu Xie, Shuchen Xue, Zijin Feng, Tianyang Hu, Jiacheng Sun, Zhenguo Li, and Cheng Zhang. Variational autoencoding discrete diffusion with enhanced dimensional correlations modeling. *arXiv preprint arXiv:2505.17384*, 2026.

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. In *ICLR*, 2025.

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.

Shaorong Zhang, Longxuan Yu, Rob Brekelmans, Luhan Tang, Salman Asif, and Greg Ver Steeg. Generation order and parallel decoding in masked diffusion models: An information-theoretic perspective. *arXiv preprint arXiv:2602.00286*, 2026.

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. In *ICLR*, 2025.

Lin Zheng, Jianbo Yuan, Lei Yu, and Lingpeng Kong. A reparameterized discrete diffusion model for text generation. In *Conference on Language Modeling*, 2024.

APPENDIX

A NORMALIZATION CONSTRAINTS

In Sec. 4, we introduced normalization constraints for both decompositions. Here, we prove that (i) these constraints ensure $p_\theta(\boldsymbol{x}|\boldsymbol{x}_t)$ is a valid probability mass function (*i.e.,* sums to 1 over V^N), and (ii) they do not reduce expressibility: any normalized tensor representable by the given decomposition is also representable under the constraints. We first prove the claim for Canonical Polyadic Decomposition (CPD), then for Tensor Train Decomposition (TTD).

A.1 CANONICAL POLYADIC DECOMPOSITION (CPD)

For CPD, recall that the constraints are given as:

$$\sum_{\alpha=1}^{r} w_\theta^{(\alpha)}(\boldsymbol{x}_t) = 1, \qquad \sum_{x^i=1}^{V} p_\theta^{(\alpha,i)}(x^i|\boldsymbol{x}_t) = 1. \tag{22}$$

We check (i) by:

$$\sum_{\boldsymbol{x}} p_\theta(\boldsymbol{x}|\boldsymbol{x}_t) = \sum_{\boldsymbol{x}} \sum_{\alpha=1}^{r} w_\theta^{(\alpha)}(\boldsymbol{x}_t) \prod_{i=1}^{N} p_\theta^{(\alpha,i)}(x^i|\boldsymbol{x}_t) = \sum_{\alpha=1}^{r} w_\theta^{(\alpha)}(\boldsymbol{x}_t) \prod_{i=1}^{N} \sum_{x^i=1}^{V} p_\theta^{(\alpha,i)}(x^i|\boldsymbol{x}_t) = \sum_{\alpha=1}^{r} w_\theta^{(\alpha)}(\boldsymbol{x}_t) \prod_{i=1}^{N} 1 = 1. \tag{23}$$

We prove (ii) by starting with a representation of a normalized tensor, then converting it to a constraint-satisfying representation of the same tensor. Given $w_\theta^{(\alpha)}(\boldsymbol{x}_t)$ and $p_\theta^{(\alpha,i)}(x^i|\boldsymbol{x}_t)$, for every α, we divide each $p_\theta^{(\alpha,i)}(x^i|\boldsymbol{x}_t)$ by $\sum_{x^i=1}^{V} p_\theta^{(\alpha,i)}(x^i|\boldsymbol{x}_t)$, and multiply $w_\theta^{(\alpha)}(\boldsymbol{x}_t)$ by the same amount, keeping the tensor unchanged. Then we have $\sum_{x^i=1}^{V} p_\theta^{(\alpha,i)}(x^i|\boldsymbol{x}_t) = 1$. And by

$$1 = \sum_{\boldsymbol{x}} p(\boldsymbol{x}) = \sum_{\alpha=1}^{r} w_\theta^{(\alpha)}(\boldsymbol{x}_t) \prod_{i=1}^{N} \sum_{x^i=1}^{V} p_\theta^{(\alpha,i)}(x^i|\boldsymbol{x}_t) = \sum_{\alpha=1}^{r} w_\theta^{(\alpha)}(\boldsymbol{x}_t)$$

we get $\sum_{\alpha=1}^{r} w_\theta^{(\alpha)}(\boldsymbol{x}_t) = 1$. It represents the same tensor, and satisfies the constraints.

A.2 TENSOR-TRAIN DECOMPOSITION (TTD)

For TTD, recall that the constraint is given as:

$$\sum_{x^i,k} \boldsymbol{G}_{i,\theta}(x^i, j, k|\boldsymbol{x}_t) = 1, \quad \forall i, j. \tag{24}$$

Similar to our proof for CPD, we begin by verifying (i):

$$\sum_{\boldsymbol{x}} p_\theta(\boldsymbol{x}|\boldsymbol{x}_t) = \sum_{\boldsymbol{x}} \tfrac{1}{r} \mathbf{1}_r^T \boldsymbol{G}_{1,\theta}(x^1|\boldsymbol{x}_t) \boldsymbol{G}_{2,\theta}(x^2|\boldsymbol{x}_t) ... \boldsymbol{G}_{N,\theta}(x^N|\boldsymbol{x}_t) \mathbf{1}_r = \tfrac{1}{r} \mathbf{1}_r^T \sum_{x^1=1}^{V} \boldsymbol{G}_{1,\theta}(x^1|\boldsymbol{x}_t) ... \sum_{x^N=1}^{V} \boldsymbol{G}_{N,\theta}(x^N|\boldsymbol{x}_t) \mathbf{1}_r. \tag{25}$$

Since by constraint $\sum_{x^i=1}^{V} \boldsymbol{G}_{i,\theta}(x^i|\boldsymbol{x}_t) \mathbf{1}_r = \mathbf{1}_r$, the summations collapse:

$$\sum_{\boldsymbol{x}} p_\theta(\boldsymbol{x}|\boldsymbol{x}_t) = \tfrac{1}{r} \mathbf{1}_r^T \sum_{x^1=1}^{V} \boldsymbol{G}_{1,\theta}(x^1|\boldsymbol{x}_t) ... \sum_{x^{N-1}=1}^{V} \boldsymbol{G}_{N-1,\theta}(x^{N-1}|\boldsymbol{x}_t) \mathbf{1}_r = ... = \tfrac{1}{r} \mathbf{1}_r^T \mathbf{1}_r = 1 \tag{26}$$

We prove (ii) in the same way as CPD. Given the cores for a normalized tensor, for position i, for every j, we divide each $\boldsymbol{G}_{i,\theta}(x^i, j, k|\boldsymbol{x}_t)$ by $\sum_{x^i,k} \boldsymbol{G}_{i,\theta}(x^i, j, k|\boldsymbol{x}_t)$, and multiply the same term to each $\boldsymbol{G}_{i-1,\theta}(x^{i-1}, l, j|\boldsymbol{x}_t)$, to keep the tensor unchanged. After this, each $\boldsymbol{G}_{i,\theta}(x^i|\boldsymbol{x}_t)$ satisfies the constraints at the cost of disrupting $\boldsymbol{G}_{i-1,\theta}(x^{i-1}|\boldsymbol{x}_t)$. This is applied for $i = N, \ldots, 2$ in decreasing order, leaving all but $\boldsymbol{G}_{1,\theta}(x^1|\boldsymbol{x}_t)$ well-constrained. For $\boldsymbol{G}_{1,\theta}(x^1|\boldsymbol{x}_t)$, we set $\boldsymbol{G}_{1,\theta}(x^1, l, j|\boldsymbol{x}_t) =$

$\frac{1}{r}\sum_l \boldsymbol{G}_{1,\theta}(x^1, l, j|\boldsymbol{x}_t)$, i.e. we run in-place column-wise mean for each matrix. This preserves the tensor, since $\mathbf{1}_r^T \boldsymbol{G}_{1,\theta}(x^1|\boldsymbol{x}_t)$ remains the same. Also it gives the final constraint, because

$$1 = \sum_{\boldsymbol{x}} p_\theta(\boldsymbol{x}|\boldsymbol{x}_t) = \tfrac{1}{r}\mathbf{1}_r^T \sum_{x^1=1}^{V} \boldsymbol{G}_{1,\theta}(x^1|\boldsymbol{x}_t) ... \sum_{x^N=1}^{V} \boldsymbol{G}_{N,\theta}(x^N|\boldsymbol{x}_t)\mathbf{1}_r = \tfrac{1}{r}\mathbf{1}_r^T \sum_{x^1=1}^{V} \boldsymbol{G}_{1,\theta}(x^1|\boldsymbol{x}_t)\mathbf{1}_r$$

is equivalent to $\sum_{x^1,j,k} \boldsymbol{G}_{1,\theta}(x^1, j, k|\boldsymbol{x}_t) = r$, thus after running the column-wise mean $\sum_{x^1,k} \boldsymbol{G}_{1,\theta}(x^1, j, k|\boldsymbol{x}_t) = 1$ for every j. It satisfies all constraints.

B An Example Demonstrating TTD's Advantage

As claimed in Sec. 4.2, we provide a simple distribution with neighbor-only dependence that can be modeled with a low TT-rank but not with a low CP-rank. Consider a tensor

$$\boldsymbol{P} = \underbrace{I_2 \otimes I_2 \otimes ... \otimes I_2}_{N-\text{times}} / 2^N,$$

representing a PMF defined over $2N$-sequences consisting of binary tokens where the paired neighbors share identical values (*e.g.,* 1100111100). We show that $\boldsymbol{P}$ has $r_{\text{CPD}} = 2^N$, but $r_{\text{TTD}} = 2$.

We first show that $r_{\text{CPD}} \leq 2^N$. Note that $\boldsymbol{P}$ can be written as a sum of scaled Dirac deltas, with each indicating a possible sequence, while being rank-1 tensors. There are 2^N of these, so the rank is at most 2^N.

Next, we show that $r_{\text{CPD}} \geq 2^N$ by using the technique of matrix flattening. Sending the odd indices to the row, and even indices to the column, and we obtain a matrix of size $2^N \times 2^N$. From the definition of $\boldsymbol{P}$, this becomes the scaled identity matrix. This matrix has rank 2^N, and since the tensor rank is always greater than or equal to the matrix rank of any of its flattenings, we get $r_{\text{CPD}} \geq 2^N$.

On the other hand $r_{\text{TTD}} \leq 2$: We construct an explicit representation with $r_{\text{TTD}} = 2$:

$$\boldsymbol{G}_{2k-1}(0) = \begin{bmatrix} 1/2 & 0 \\ 1/2 & 0 \end{bmatrix}, \boldsymbol{G}_{2k-1}(1) = \begin{bmatrix} 0 & 1/2 \\ 0 & 1/2 \end{bmatrix}$$

$$\boldsymbol{G}_{2k}(0) = \begin{bmatrix} 1/2 & 1/2 \\ 0 & 0 \end{bmatrix}, \boldsymbol{G}_{2k}(1) = \begin{bmatrix} 0 & 0 \\ 1/2 & 1/2 \end{bmatrix}$$

then

$$\boldsymbol{P}(\nu_1, \ldots, \nu_{2N}) = \tfrac{1}{r}\mathbf{1}_r^T \boldsymbol{G}_1(\nu_1)\boldsymbol{G}_2(\nu_2)...\boldsymbol{G}_{2N-1}(\nu_{2N-1})\boldsymbol{G}_{2N}(\nu_{2N})\mathbf{1}_r$$

$r_{\text{TTD}} \geq 2$: $r_{\text{TTD}} = 1$ is equivalent to being a rank-1 tensor, which is not the case since $r_{\text{CPD}} \geq 2^N$.

Thus $r_{\text{CPD}} = 2^N, r_{\text{TTD}} = 2$. □

C A Pitfall of RoPE in Masked Diffusion

Rotary Position Embedding (RoPE) (Su et al., 2024) incorporates positional information into the attention layer by "rotating" queries and keys by

$$q_m = R^d_{\Theta,m} W_q x_m, k_n = R^d_{\Theta,n} W_k x_n$$

where $R^d_{\Theta,m}$ and $R^d_{\Theta,n}$ are appropriately defined rotary matrices. (Here we write position indices in subscripts to match Su et al. (2024).)

This method, however, should be used with care on MDMs. In the very first step of MDM generation, the input is a fully-masked sequence. Since default RoPE neither changes the token embeddings nor the values in attention layers, it loses its ability to distinguish between positions.

If the initial $x_m = x$ is uniform across position, $v_m = W_v x_m = W_v x = v$ is also uniform, and thus the output of attention layer

$$o_m = \sum_{n=1}^{N} a_{m,n} v_n = v \sum_{n=1}^{N} a_{m,n} = v \tag{27}$$

also becomes uniform regardless of the attention weights a. All the other components, such as normalizing layers or dense layers, are applied independently to each position, maintaining this unintended uniformity. Indeed, we confirm that under this formulation, the outputted logits of the model are identical for every position in the first step.

Although this issue is less pronounced in settings such as conditional generation (where the first step is not fully masked), long sequence generation (where the impact of an incorrectly chosen first token is negligible), or datasets where true marginal probabilities are similar across positions, we observed noticeable performance degradation in our QM9 (Blum & Reymond, 2009; Rupp et al., 2012) experiments, where the model performs unconditional generation of short sequences. This motivated a fix: applying RoPE rotary matrices to values in addition to queries and keys in attention. Note that we retain the previous behavior in Sec. 5.1 for compatibility with the pretrained model.

D Experiment Details

D.1 Text Generation

For OWT (Gokaslan et al., 2019) and LM1B (Chelba et al., 2014), we matched the training time for each method. Each resulted in a total training time of approximately 10 hours on four NVIDIA A6000 GPUs.

For MDLM (Sahoo et al., 2024) we fine-tuned the base model with global batch size 64, constant learning rate 1×10^{-5} with Adam for 100k steps, and for CPD and TTD we used global batch size 4 for 150k steps.

Di4C (Hayakawa et al., 2025) model was trained for 13k steps with the base model as the teacher model, with other hyperparameters set to default values in the released code of global batch size 4 and constant learning rate 3×10^{-5} with warmup for 2500 steps with AdamW (Loshchilov & Hutter, 2019) optimizer, alongside exponential moving average (EMA) with decay rate of 0.9999.

For EDLM (Xu et al., 2025), we used the checkpoints from MDLM with the AR-based method, and used the default sampling parameter of sampling size $k = 2$ but widened the sampling window to $w = 1$ to follow the setting outlined in their paper.

For DCD (Liu et al., 2025), we followed the released code and used sedd-medium (Lou et al., 2024) as the diffusion model and gpt2 (Radford et al., 2019) as the AR model with a chunk size of 1 to simulate fully random generation.

For VADD (Xie et al., 2026), we trained the base model with the default values in the released code of global batch size 512 and constant learning rate 3×10^{-4} with warmup for 2500 steps with AdamW and EMA with decay rate 0.9999 for 5k steps on latent dimension of 512. VADD and our methods were fine-tuned from the base model in the same setting, with VADD for 1k steps and our methods for 400 steps. For sampling we followed their default setting using fixed latent.

For measuring the sampling times, we generated 64 sequences with 128 timesteps on batch size of 1 for every method and computed the sampling time of a single sequence. Sampling time measurements were conducted on a single NVIDIA PRO6000 GPU.

D.2 Molecule Generation

The base MDLM (Sahoo et al., 2024) was trained with global batch size 8192, constant learning rate 1×10^{-4} with Adam (Kingma & Ba, 2015) for 50k steps. For MDLM and our methods, we fine-tuned the base model under the same settings for 10k steps.

Di4C (Hayakawa et al., 2025) model was trained for 10k steps with our base model as the teacher model, with global batch size 512 and other hyperparameters set to default values in the released

Table 5: Negative log-likelihood on masked validation set for different permutations of OpenWebText (Gokaslan et al., 2019).

NLL (↓)	Identity	Stride 2	Stride 4	Stride 32	Stride 256	Random
Marginal	6.83	6.86	6.98	6.98	6.87	7.42
TTD	6.32	6.66	6.88	6.86	6.82	7.37
Gap	**-0.51**	-0.21	-0.10	-0.11	-0.05	-0.06

code of constant learning rate 3×10^{-5} with warmup for 2500 steps with AdamW (Loshchilov & Hutter, 2019) and exponential moving average (EMA) with decay rate 0.9999.

VADD (Xie et al., 2026) model was trained for 60k steps from scratch with global batch size 2048 and other hyperparameters set to default values in the released code of constant learning rate 3×10^{-4} with warmup for 2500 steps with AdamW and EMA with decay rate 0.9999 on latent dimension of 512. For sampling of VADD we also followed their default setting using fixed latent.

Experiments were conducted using four GPUs of either NVIDIA A6000 or PRO6000.

E Additional Results

E.1 Text Generation

In Sec. 5.1, we have claimed that TTD's outperformance is due to its ability to efficiently model local dependencies. We provide further evidence for this by constructing a dataset of different permutations of the OpenWebText (Gokaslan et al., 2019) dataset, training a small model from scratch, and measuring the negative log-likelihood (NLL) on samples with mask ratio 0.9 in the validation set. The permutations include the identity permutation (e.g. 1 2 3 4 5 6 $\rightarrow$ 1 2 3 4 5 6), stride permutations where the sequence is chunked and interleaved, such that the neighboring tokens get seperated in a way that RoPE can easily model (e.g. stride 2, 1 2 3 4 5 6 $\rightarrow$ 1 4 2 5 3 6), and random permutation of fixed ordering.

The results of Tab. 5 show that the advantage of TTD is indeed greatest for the identity permutation where the dependency between nearby token is strongest, and diminishes as the stride (and therefore the average dependency length) increases.

E.2 Molecule Generation

In Fig. 4 where we generate molecules with different ordering strategies, we observe that the gap between TTD and CPD is most clearly pronounced in left-to-right order generation. This agrees with our theory that TTD outperforms CPD due to its ability to model the dependence between neighboring tokens. Orthogonal to the ordering strategy, we also adapt EB-Sampler (Ben-Hamu et al., 2025), which adaptively determines the number of unmasked *tokens* to validate whether our models can be paired with advanced samplers. As shown in Fig. 5, we observe that our methods outperform the baseline even with a different choice of sampler.

From Fig. 6, we can see that the performance of TTD monotonically increases with the number of ranks used, as expected.

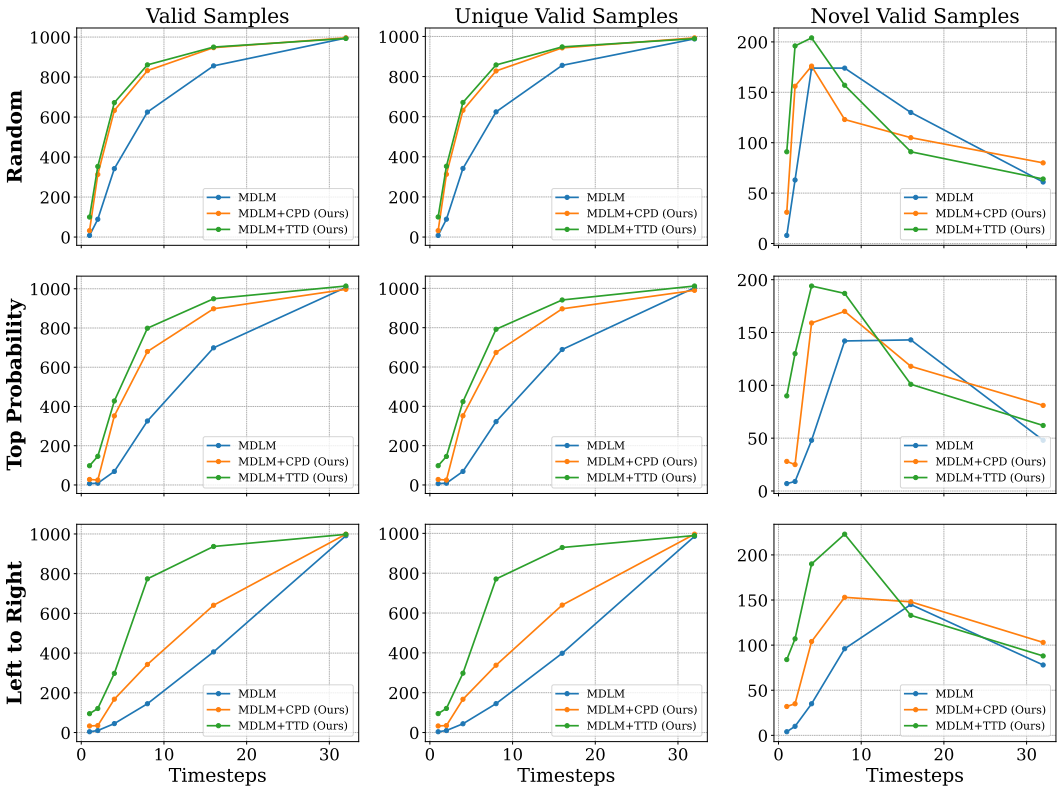


Figure 4: Molecule generation on QM9 (Blum & Reymond, 2009; Rupp et al., 2012) for different sampling strategies. Each row corresponds to random order, top-probability-based order and left-to-right order.

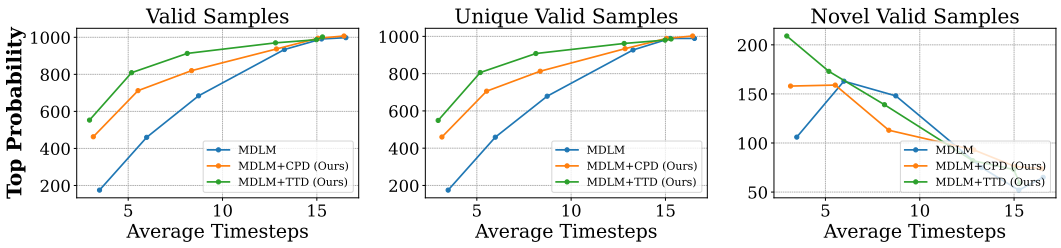


Figure 5: Molecule generation on QM9 (Blum & Reymond, 2009; Rupp et al., 2012) with EB-Sampler (Ben-Hamu et al., 2025). "Average Timesteps" indicates the average number of timesteps taken by the sampler to generate samples.

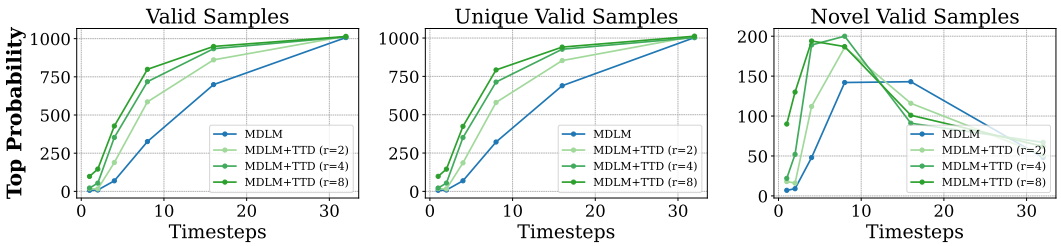


Figure 6: Rank size ablation on QM9 (Blum & Reymond, 2009; Rupp et al., 2012)