Auto-Regressive Masked Diffusion Models

Mahdi Karami[1,2*] **Ali Ghodsi**[1]
[1] School of Computer Science, University of Waterloo, ON, Canada
[2] Google Research
[*] Correspondence to: mahdika@google.com

Abstract

Masked diffusion models (MDMs) have emerged as a promising approach for language modeling, yet they face a performance gap compared to autoregressive models (ARMs) and require more training iterations. In this work, we present the Auto-Regressive Masked Diffusion (ARMD) model, an architecture designed to close this gap by unifying the training efficiency of autoregressive models with the parallel generation capabilities of diffusion-based models. Our key insight is to reframe the masked diffusion process as a block-wise causal model. This perspective allows us to design a strictly causal, permutation-equivariant architecture that computes all conditional probabilities across multiple denoising steps in a single, parallel forward pass. The resulting architecture supports efficient, autoregressive-style decoding and a progressive permutation training scheme, allowing the model to learn both canonical left-to-right and random token orderings. Leveraging this flexibility, we introduce a novel *strided parallel generation* strategy that accelerates inference by generating tokens in parallel streams while maintaining global coherence. Empirical results demonstrate that ARMD achieves state-of-the-art performance on standard language modeling benchmarks, outperforming established diffusion baselines while requiring significantly fewer training steps. Furthermore, it establishes a new benchmark for parallel text generation, effectively bridging the performance gap between parallel and sequential decoding.

Proceedings of the 29[th] International Conference on Artificial Intelligence and Statistics (AISTATS) 2026, Tangier, Morocco. PMLR: Volume 300.

1 INTRODUCTION

Autoregressive Models (ARMs) have demonstrated significant success across various sequence modeling tasks, particularly in language modeling with prominent models like GPT (OpenAI, 2023) and LLaMA (Touvron et al., 2023). The foundational principle of ARMs lies in factorizing the joint probability distribution of a sequence into a product of conditional probabilities using the chain rule (Larochelle and Murray, 2011; Bengio and Bengio, 2000), typically following a fixed, left-to-right generation order. This enables efficient parallel training and has achieved state-of-the-art performance in diverse domains beyond text, such as image generation (Van Den Oord et al., 2016a), audio synthesis (Van Den Oord et al., 2016b), and graph modeling (Liao et al., 2019; Karami, 2024). Despite their success, ARMs suffer from the inherent sequential nature of their generation process, leading to slow sampling, especially for long sequences, and challenges for tasks requiring bidirectional context or non-sequential reasoning (Berglund et al., 2023; Ding et al., 2023; Bachmann and Nagarajan, 2024).

In parallel, diffusion models have emerged as a powerful generative modeling alternative, achieving remarkable success in continuous domains such as image generation (Sohl-Dickstein et al., 2015; Ho et al., 2020; Song and Ermon, 2019). A key advantage of these models is their ability to perform parallel sampling during inference, which, combined with their high degree of controllability, makes them well-suited for conditional and guided generation tasks. This potential has motivated a renewed focus on extending diffusion principles to discrete data modalities (Austin et al., 2021; Lou et al., 2023), including text (Gulrajani and Hashimoto, 2023; Ou et al., 2024; Lou et al., 2023; Nie et al., 2025) and graph (Vignac et al., 2022) and biological sequences (Sahoo et al., 2024).

Despite these advancements, discrete diffusion models have struggled to match the performance benchmarks set by ARMs in language modeling. Moreover, they

typically require longer training schedules and a large number of denoising steps during inference, which can severely hinder their sampling efficiency (Austin et al., 2021; Lou et al., 2023). Addressing these limitations has become an active area of research, with recent efforts aiming to improve both the scalability and expressiveness of discrete diffusion models.

Contributions. To address these challenges, we introduce the *Auto-Regressive Masked Diffusion (ARMD)* model, which unifies the strengths of diffusion-based learning with the efficiency of autoregressive models. First, we reframe masked diffusion models (MDMs) as block-wise causal models. This perspective enables the parallel evaluation of all conditional probabilities within a sequence using a single network call, significantly improving training efficiency. Leveraging this insight, we propose a causal, permutation-equivariant, attention-based architecture that generalizes traditional autoregressive models. This flexible design supports hybrid training, allowing the model to learn effectively from both canonical left-to-right and random orderings. ARMD is also compatible with key-value caching, enabling efficient, autoregressive-style decoding during inference. Furthermore, we introduce a novel *strided parallel generation* strategy that accelerates sampling by generating tokens in parallel streams and bridges the performance gap with sequential generation. The resulting model achieves state-of-the-art performance across standard language modeling benchmarks, while requiring significantly fewer training steps than competing diffusion-based methods.

2 BACKGROUND

Diffusion Generative Models. Diffusion models represent a class of probabilistic generative models that learn a data distribution by systematically reversing a process that corrupts data with noise. These models consist of two core components: a *forward diffusion process* that progressively adds noise to a clean data sample $\boldsymbol{z}^0 \sim p_{data}(\boldsymbol{z}^0)$, and a reverse *denoising process*, which is trained to invert this corruption. In the continuous domain, the forward process typically adds Gaussian noise over a chain of T timesteps, producing a series of increasingly noisy latent variables $[\boldsymbol{z}^1, \boldsymbol{z}^2, \dots, \boldsymbol{z}^T]$, where $\boldsymbol{z}^T$ approximates a simple prior distribution, such as an isotropic Gaussian. The forward process is defined as a Markov chain specified by a predefined conditional probability $q(\boldsymbol{z}^t \mid \boldsymbol{z}^{t-1})$. Consequently, the joint distribution of the latent variables can be factorized as: $q(\boldsymbol{z}^1, \boldsymbol{z}^2, \dots, \boldsymbol{z}^T \mid \boldsymbol{z}^0) = \prod_{t=1}^{T} q(\boldsymbol{z}^t \mid \boldsymbol{z}^{t-1})$, For sufficiently large T, $q(\boldsymbol{z}^T)$ gets close to a noise prior distribution. The backward denoising process, parameterized by a neural network $p_\theta(\boldsymbol{z}^{t-1} \mid \boldsymbol{z}^t, t)$, is trained to approximate the time-reversed dynamics of this diffusion chain and generate clean samples from the data distribution.

In *discrete domains*, such as text, where data points belong to a finite set of K categories, each sample is represented by a one-hot vector $\boldsymbol{z}^0 \in \{0,1\}^K$. Following the framework proposed by Austin et al. (2021), the discrete forward process can be modeled using a sequence of stochastic transition matrices $\{\boldsymbol{Q}^t\}_{t=1}^T$, where $[Q^t]_{ij}$ specifies the probability of transitioning from state j (the j-th class) to state i (the i-th class). The conditional distribution at time step t is thus defined by a categorical distribution: $q(\boldsymbol{z}^t \mid \boldsymbol{z}^{t-1}) = \mathrm{Cat}(\boldsymbol{z}^t;\ \boldsymbol{Q}^t \boldsymbol{z}^{t-1})$.

A common parameterization for the transition matrix interpolates between the data and a stationary noise distribution $\boldsymbol{\phi} \in \Delta^{K-1}$ (the K-simplex): $\boldsymbol{Q}^t = \beta^t \mathbf{I} + (1-\beta^t)\boldsymbol{\phi}\mathbf{1}^\top$, where $\beta^t \in [0,1]$ controls the noise level at step t and $\boldsymbol{\phi}$ is typically a uniform distribution with $\mathbf{1}$ denoting an all-one vector. This equation leads to a closed-form expression for the transition:

$$q(\boldsymbol{z}^t \mid \boldsymbol{z}^{t-1}) = \mathrm{Cat}(\boldsymbol{z}^t;\ \beta^t \boldsymbol{z}^{t-1} + (1-\beta^t)\boldsymbol{\phi}),$$
$$q(\boldsymbol{z}^t \mid \boldsymbol{z}^0) = \mathrm{Cat}(\boldsymbol{z}^t;\ \alpha^t \boldsymbol{z}^0 + (1-\alpha^t)\boldsymbol{\phi}),$$

where $\alpha^t = \prod_{i=1}^{t} \beta^i$ is the cumulative probability that a token remains in its initial state after t steps. This interpolation between the clean data and a fixed noise distribution is analogous to additive Gaussian noise in the continuous setting (Sahoo et al., 2024).

Masked Diffusion Models. A simple yet effective variant of discrete diffusion, particularly relevant for sequence data like text, is the Masked Diffusion Model (MDM). This approach introduces an additional special state, denoted by `[mask]`, therefore $\boldsymbol{z}_t \in \{0,1\}^{K+1}$, with the $(K+1)$-th dimension corresponding to the mask token. The noise process is designed such that the `[mask]` state is absorbing: once a token is masked, it remains masked in all subsequent steps, and at step T, all tokens are fully masked. In this setting, the noise distribution $\boldsymbol{\phi}$ is replaced by a one-hot vector $\boldsymbol{m}$ with a 1 at index $K+1$. Thus, the forward transition simplifies to: $q(\boldsymbol{z}^t \mid \boldsymbol{z}^0) = \mathrm{Cat}(\boldsymbol{z}^t;\ \alpha^t \boldsymbol{z}^0 + (1-\alpha^t)\boldsymbol{m})$. MDMs have often achieved state-of-the-art performance over a range of discrete generative tasks (Austin et al., 2021; Lou et al., 2023).

The reverse process, $p_\theta(\boldsymbol{z}_j^{t-1} | \boldsymbol{z}_{1:N}^t)$, is trained to iteratively recover the clean sequence $\boldsymbol{z}_{1:N}^0$ from a fully masked input. This is achieved by optimizing the Evidence Lower Bound (ELBO) on the log-likelihood of

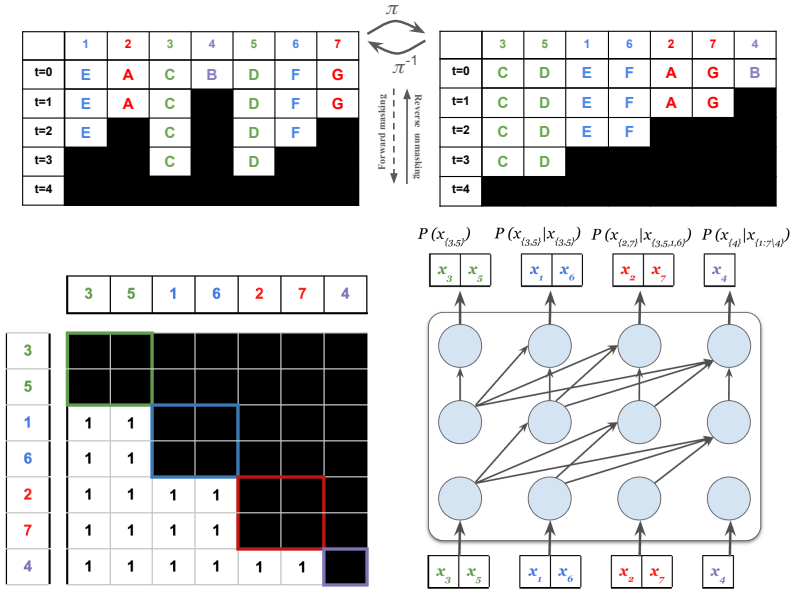

Figure 1: *(Top Left)* An instance of the masked diffusion process on $\boldsymbol{z}_{1:N}$ for $T = 4$ steps. *(Top Right)* The causal patterns derived by permuting the sequence $\boldsymbol{x}_{1:N} = \pi(\boldsymbol{z}_{1:N})$. The input sequence is partitioned into blocks according to the reverse masking order: $\mathcal{X}(1) = \{3, 4\}$, $\mathcal{X}(2) = \{1, 6\}$, $\mathcal{X}(3) = \{2, 7\}$, and $\mathcal{X}(4) = \{4\}$. *(Bottom Left)* The resulting strictly causal attention mask. *(Bottom Right)* A model instance composed of a single strictly causal layer followed by two causal layers, forming a deep strictly causal architecture.

1. The _ _ _ _ _ _ _
2. The _ _ _ jumps _ _ _
3. The quick _ _ jumps over _ _
4. The quick brown _ jumps over the _
5. The quick brown fox jumps over the dog

Figure 2: Visualization of *strided parallel generation*. Steps 1–2 represent the sequential generation of the stream heads. Steps 3–5 illustrate the parallel generation of subsequent tokens, where tokens of the same color are generated simultaneously. Figure 4 depicts the corresponding diffusion process and attention mask.

the data (Austin et al., 2021):

$$-\log p_\theta(\boldsymbol{z}^0) \le \mathcal{L}_{\text{prior}} + \mathcal{L}_0 + \underbrace{\sum_{t=1}^{T} \mathbb{E}_q \left[D_{\text{KL}} \left(q(\boldsymbol{z}^{t-1} | \boldsymbol{z}^0, \boldsymbol{z}^t) \,\|\, p_\theta(\boldsymbol{z}^{t-1} \mid \boldsymbol{z}^t_{1:N}) \right) \right]}_{\mathcal{L}_{\text{diff}}}$$

where D_{KL} is the Kullback–Leibler divergence, $\mathcal{L}_0 = \mathbb{E}_{q(\boldsymbol{z}^0)} \left[-\log p_\theta(\boldsymbol{z}^0 | \boldsymbol{z}^1) \right]$ is the reconstruction loss for the final denoising step and $\mathcal{L}_{\text{prior}} = D_{\text{KL}} \left(q(\boldsymbol{z}^T \mid \boldsymbol{z}^0) \,\|\, p(\boldsymbol{z}^T) \right)$ measures the discrepancy between the forward process's final distribution and the noise prior. For MDMs where the forward process deterministically leads to a fully masked state, $q(\boldsymbol{z}^T \mid \boldsymbol{z}^0) = p(\boldsymbol{z}^T) = \boldsymbol{m}$, then $\mathcal{L}_{\text{prior}} = 0$, and the discrete time loss simplifies to (Sahoo et al., 2024):

$$\mathcal{L}_{\text{diff}} = \sum_{t=1}^{T} \mathbb{E}_q \left[\frac{\alpha^t - \alpha^{t-1}}{1 - \alpha^t} \underbrace{\sum_{\{j | \boldsymbol{z}^t_j = \boldsymbol{m}\}} \log p_\theta(\boldsymbol{z}^{t-1}_j \mid \boldsymbol{z}^t_{1:N})}_{L_t} \right] \tag{1}$$

Here, the inner loss term L_t is a cross entropy objective computed specifically on the subset of positions that were just masked at step t, thereby focusing the prediction only on unmasking them.

Autoregressive Models (ARMs), on the other hand, factorize the joint probability of a sequence into a product of univariate conditionals using the chain rule. Given a predefined ordering of the variables (e.g., left-to-right), the log-likelihood is decomposed as:

$$\log p(\boldsymbol{x}_{1:N}) = \sum_{t=1}^{N} \log p(\boldsymbol{x}_t \mid \boldsymbol{x}_{<t}),$$

where each token $\boldsymbol{x}_t$ is predicted based on the preceding tokens in the sequence ($x_{<t}$). Thanks to causal masking and teacher forcing, ARMs can evaluate the full log-likelihood for a sequence in a single forward pass, allowing for highly parallelized and efficient training. In contrast, the training objective for MDMs (1) requires sampling a different timestep t for each training example and making a prediction with the model. To optimize the full objective, this requires many neural network calls, each evaluating a portion of likelihood corresponding to a denoising step, L_t, making its training less efficient per epoch.

However, the MDM training process can be reformulated to parallelize its objective. The key insight is that the random masking schedule induces a causal structure by reordering the sequence. Formally, for a sample from masking process, let $\tau(j) \in \{1, ..., T\}$ denote the *masking timestep* at which the j-th token of the original sequence $\boldsymbol{z}_{1:N}$ is changed to `[mask]`. We define a permutation π over the token indices that sorts the tokens based on their masking times in descending order:

$\tau(\pi(1)) \geq \tau(\pi(2)) \geq ... \geq \tau(\pi(N))$. This reorders the sequence so that tokens masked last appear first in the new ordering. Let the resulting permuted sequence be denoted by $\boldsymbol{x}_{1:N} = \pi(\boldsymbol{z}_{1:N}) = (\boldsymbol{z}_{\pi(1)}, \boldsymbol{z}_{\pi(2)}, ..., \boldsymbol{z}_{\pi(N)})$. This reordering induces a partitioning of $\boldsymbol{x}_{1:N}$ into T non-overlapping (and possibly variable-length) blocks: $\boldsymbol{X} = [\mathcal{X}^{(1)}, \mathcal{X}^{(2)}, \ldots, \mathcal{X}^{(T)}]$, sorted by their generative (unmasking) order. Each block $\mathcal{X}^{(b)} \in \mathbb{R}^{n_b \times d}$ contains n_b tokens, and $\sum_{b=1}^{T} n_b = N$. The block index of token n, denoted by $\mathcal{B}(n) \in \{1, \ldots, T\}$, is then $\mathcal{B}(n) := T - \tau(n) + 1$. As illustrated in Figure 1, this reordering naturally reveals *a block-wise causal* (block triangular) structure: predicting the tokens in a block $\mathcal{X}^{(t)}$ is conditionally dependent only on tokens in preceding blocks $\mathcal{X}^{(<t)}$. This allows the diffusion loss to be represented in an autoregressive style:

$$\mathcal{L}_{\text{diff}} = \mathbb{E}_q \Big[\sum_{t=1}^{T-1} \gamma(t) \sum_{n \in \mathcal{B}(t)} - \log p_\theta(\boldsymbol{x}_n \mid [\boldsymbol{x}_i \mid \mathcal{B}(i) \leq T - t]) \Big] \tag{2}$$

where $\gamma(t)$ is the reweighting factor that reflects the diffusion schedule. This reformulation of the masked diffusion process as a causal generative model reveals a conceptual and structural connection between MDMs and ARMs, enabling parallel evaluation of all conditionals in the loss by a single forward pass. Importantly, the conditional probabilities in the diffusion objective (1) are invariant to the permutation of the evidence set. Additionally, L_t is invariant to the ordering of tokens within the target block $\mathcal{B}(t)$. Therefore, this structure necessitates a *permutation-equivariant* sequence model to parameterize these conditionals.

In the following section, we introduce a deep, permutation-equivariant, and strictly causal sequence model tailored for efficient training of masked diffusion models. Unlike MLM or BERT models that represent masking by inserting explicit `[mask]` tokens into the input, the proposed architecture controls information flow via a carefully designed attention mask. This mechanism restricts output representations from accessing input positions designated as masked, directly enforcing the conditional dependencies required by the denoising process.

3 STRICTLY CAUSAL MODELS

As explained in the previous section and in equation 2, the conditional probability is parameterized by a sequence model, where the output at any given position n depends exclusively on preceding input blocks—*i.e.* the present and all future blocks are masked so they cannot influence the current output. We call any network satisfying this property *strictly causal* which is illustrated in Figure 1. Formally, a *strictly causal layer* is defined as $y_n = f^{sc}(x_{<\mathcal{B}(n)})$. This is distinct from a standard *causal layer*, where the output can also depend on the current input block, i.e. $y_n = f^{c}(x_{\leq \mathcal{B}(n)})$.

Strictly causal systems form a proper subclass of causal systems which are also physically realizable. Importantly, composing a strictly causal layer with any number of causal ones (e.g., $f^c \circ f^{sc}$) yields a network that preserves overall strict causality. This property allows us to construct arbitrarily deep, strictly causal networks by stacking arbitrarily many causal layers and interleaving just one strictly causal layer (block) which is particularly advantageous in designing our deep neural network architectures.[1]

Causal Self-Attention. To implement this principle in a modern transformer architecture, we adapt the standard self-attention mechanism. The output of a standard self-attention layer for an input sequence $\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N]^\top \in \mathbb{R}^{N \times d}$ (where $\boldsymbol{x}_n \in \mathbb{R}^d$) is given by:

$$\boldsymbol{Y} = \text{SA}(\boldsymbol{X}) = \text{Softmax}(\boldsymbol{Q}\,\boldsymbol{K}^\top)\,\boldsymbol{V}$$

Here, the query, key, and value matrices are linear projections of the input: $\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}^q$, $\boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}^k$, and $\boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}^v$, where $\boldsymbol{W}^q, \boldsymbol{W}^k, \boldsymbol{W}^v \in \mathbb{R}^{d \times d}$. By design, this operation is permutation-equivariant. Defining attention scores corresponding to token n as $A_{(n,i)} = \text{Softmax}_i(\boldsymbol{q}_n^\top \boldsymbol{K}^\top)$, the output at time n can be written as a weighted sum over the entire sequence: $\boldsymbol{y}_n = \text{SA}_n(\boldsymbol{X}) = \sum_{i=1}^{N} A_{(n,i)} \boldsymbol{v}_i$.

To enforce causality at the block level, we partition the input sequence $\boldsymbol{X}$ into T non-overlapping blocks, $\boldsymbol{X} = [\mathcal{X}^{(1)}, \mathcal{X}^{(2)}, \ldots, \mathcal{X}^{(T)}]$, and introduce a block-wise causal mask, $\boldsymbol{M}^c$. This mask modifies information flow and restricts each token to attend only to tokens in the current or preceding blocks:

$$\begin{aligned} \boldsymbol{y}_n &= \text{SA}^c\left([\boldsymbol{x}_i \mid \mathcal{B}(i) \leq \mathcal{B}(n)]\right)[n] \\ &= \sum_{i:\ \mathcal{B}(i) \leq \mathcal{B}(n)} \text{Softmax}_i(\boldsymbol{q}_n^\top \boldsymbol{K}^\top + M^c_{(n,i)})\,\boldsymbol{v}_i, \end{aligned} \tag{3}$$

where, the mapping $\mathcal{B}(n) \in \{1, \ldots, T\}$ returns the block index of token n. The output sequence in matrix form is then:

$$\boldsymbol{Y} = \text{SA}^c(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}; \boldsymbol{M}^c) = \text{Softmax}(\boldsymbol{Q}\,\boldsymbol{K}^\top + \boldsymbol{M}^c)\,\boldsymbol{V}$$

where $\boldsymbol{M}^c \in \{0, -\infty\}^{N \times N}$ is the causal mask matrix [2]. This results in a block-lower-triangular attention

[1] It's important to note that within our framework, causality is defined by the processing order of information (or absorbing noise), not necessarily the temporal order of tokens in the original sequence.

[2] Refer to Appendix A for a detailed notation definition.

matrix, allowing full attention within each block (intra-block) while prohibiting attention to any future blocks.

In this block-wise causal attention, the output $\boldsymbol{y}_n$ for token n is generated through the interaction of its corresponding query vector $\boldsymbol{q}_n$ with the key and value context of its current block and any preceding blocks. Hence, the output $\boldsymbol{y}_n$ depends on the current block, $\mathcal{B}(n)$, in two ways: 1) through its own query vector $\boldsymbol{q}_n$, and 2) by attending to the key-value cache within the same block (*i.e.* $\{\boldsymbol{k}_i, \boldsymbol{v}_i \mid i \in \mathcal{B}(n)\}$. This allows for a full intra-block interaction.

To construct a *strictly causal attention layer*, we must completely eliminate any dependence on the current block. This involves two key modifications to prevent any information leakage from the current block. First, we apply a strictly causal attention mask, $\boldsymbol{M}^{sc}$, that permits attention only to keys/values from preceding blocks. Second, we employ a representation for the query that depends only on the past blocks (instead of the standard query $\boldsymbol{q}_n$ which is a function of the current input $\boldsymbol{x}_n$). Specifically, we define a modified query as a strictly causal function:

$$\hat{\boldsymbol{q}}_n = f_q^{sc}\left(\{\boldsymbol{q}_i \mid \mathcal{B}(i) < \mathcal{B}(n)\}\right), \tag{4}$$

where $f_q^{sc}(\cdot)$ is a permutation-equivariant aggregation function (e.g., mean-pooling or learned transformation) that pools information from the queries of previous blocks to ensure $\hat{\boldsymbol{q}}_n$ has no dependence on the current block $\mathcal{B}(i)$.

By incorporating these two constraints, the output of the *strictly causal attention* layer is defined as:

$$\begin{aligned}\boldsymbol{y}_n &= \mathrm{SA}^{sc}\left([\boldsymbol{x}_i \mid \mathcal{B}(i) < \mathcal{B}(n)]\right)[n] \\ &= \sum_{i:\mathcal{B}(i)<\mathcal{B}(n)} \mathrm{Softmax}_i\left(\hat{\boldsymbol{q}}_n^\top \boldsymbol{K}^\top + M^{sc}_{(n,i)}\right)\boldsymbol{v}_i, \end{aligned} \tag{5}$$

This formulation ensures that both the query and the context it attends to are derived strictly from inputs preceding the current block, thereby satisfying the condition for strict block-level causality. In the following, we explore different design choices for constructing the strictly causal query aggregation function, $f_q^{sc}(\cdot)$.

Shift: A straightforward approach is to define the modified query at position n using the token immediately preceding it, resulting in a simple one-step shift function: $\hat{\boldsymbol{q}}_n = f_q^{sc}(\boldsymbol{q}_{n-1})$. This operation effectively conditions the query on the most recent token from the past. While simple, it is inherently not permutation-equivariant and is only suitable for a fixed forward (left-to-right) processing order. Consequently, a deep Transformer architecture that incorporates a single attention layer with this shifted query—while other layers use standard causal attention—collapses into a conventional autoregressive language model, which learns to sequentially generate the next token in a left-to-right order.

To derive a strictly causal and permutation-equivariant query representation suitable for our problem, we explore alternative architectural designs. Our goal is to minimize extra parameters and computational overhead while maintaining high expressivity. Our strategy involves the following two main components:

Prefix Aggregation: First, our approach involves aggregating information from all past tokens (the prefix) into a summary vector that can be used to form the strictly causal query. We introduce a specialized feed-forward layer that computes an intermediate representation, $\boldsymbol{g}_n^0$, as a position-aware weighted sum of all inputs from preceding blocks:

$$\boldsymbol{g}_n^0 = \mathrm{FF}^{sc}\left([\boldsymbol{x}_i \mid \mathcal{B}(i) < \mathcal{B}(n)]\right)[n] = \sum_{i:\mathcal{B}(i)<\mathcal{B}(n)} H_{(n,i)}\,\boldsymbol{x}_i$$

where $H_{(n,i)}$ is the weight coupling an input token i to an output token n. Directly parameterizing the weight matrix $\boldsymbol{H}$ is computationally expensive, requiring a quadratic number of parameters and operations relative to the sequence length. To circumvent this, we employ an efficient parameterization where the weight $H_{(n,i)}$ is defined as a function of the actual positions in the sequence ($\hat{\pi}(i)$ and $\hat{\pi}(n)$). Specifically, we parameterize $H_{(n,i)}$ using the dot-product similarity of their corresponding positional embedding vectors:

$$H_{(n,i)} = \langle \mathbf{PE}_{\hat{\pi}(n)}, \mathbf{PE}_{\hat{\pi}(i)} \rangle = \langle \hat{\mathbf{PE}}_n, \hat{\mathbf{PE}}_i \rangle.$$

Here, $\mathbf{PE}_j$ represents the j-th row of the positional embedding matrix $\mathbf{PE} \in \mathbb{R}^{N \times d_{pe}}$, $\hat{\pi}(n)$ is the position index of nth token, and $\hat{\mathbf{PE}} = \pi(\mathbf{PE})$ denotes the positional embedding matrix reordered according to the sequence permutation π. This parameterization ensures the layer has a *fixed number of parameters*, independent of sequence length. Moreover, we can cast the computation of $\boldsymbol{g}_n^0$ as a *linear attention* form (Katharopoulos et al., 2020), the full block matrix computation can be implemented efficiently using a masking matrix to enforce strict causality as:

$$\boldsymbol{G}^0 = \mathrm{PA}^{sc}(\boldsymbol{X}) = \left((\hat{\mathbf{PE}} \times \hat{\mathbf{PE}}^\top) \odot \boldsymbol{M}^{sc}\right)\boldsymbol{X} \tag{6}$$

where $\boldsymbol{M}^{sc} \in \{0,1\}^{N \times N}$ is a strictly causal mask. The output, $\boldsymbol{g}_n^0$, subsequently serves as the basis for deriving the strictly causal query $\hat{\boldsymbol{q}}_n$. The resulting layer inherits the permutation-equivariance (with respect to the condition set) of the attention. Furthermore, this linear attention can be reformulated as a recurrent form (Katharopoulos et al., 2020), offering computational benefits: it enables sub-quadratic parallel training through techniques such as chunkwise processing (Hua et al., 2022; Kacham et al., 2024) or parallel

scan algorithms (Blelloch, 1990; Smith et al., 2023), and enjoys a constant-time complexity per generated token at inference.

Two-Stream Strictly Causal-Attention To efficiently construct a deep strictly causal architecture—without introducing additional parameters—we adopt a *two-stream attention architecture*. At each layer l, this design maintains and updates two parallel representations: a causal representation $\boldsymbol{X}^l$ and a strictly causal representation $\boldsymbol{G}^l$. The key idea is that both streams share the same attention parameters and key-value context, but operate with different masking and query inputs: the causal stream allows access to the current and past tokens, while the strictly causal stream uses the stricter mask and derives its query $\hat{\boldsymbol{Q}}$ from the previous strictly causal output $\boldsymbol{G}^{l-1}$. Formally, the update rules for layer l are:

$$\boldsymbol{X}^l = \mathrm{SA}^c_\theta(\boldsymbol{X}^{l-1}; \boldsymbol{M}^c) = \mathrm{Attn}\left(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}; \boldsymbol{M}^c\right), \tag{7}$$

$$\boldsymbol{G}^l = \mathrm{SA}^{sc}_\theta(\boldsymbol{G}^{l-1}, \boldsymbol{X}^{l-1}; \boldsymbol{M}^{sc}) = \mathrm{Attn}\left(\hat{\boldsymbol{Q}}, \boldsymbol{K}, \boldsymbol{V}; \boldsymbol{M}^{sc}\right)$$

where the query, key, and value projections are defined as: $\boldsymbol{Q} = \boldsymbol{W}^q \boldsymbol{X}^{l-1}$, $\boldsymbol{K} = \boldsymbol{W}^k \boldsymbol{X}^{l-1}$, $\boldsymbol{V} = \boldsymbol{W}^v \boldsymbol{X}^{l-1}$, $\hat{\boldsymbol{Q}} = \boldsymbol{W}^q \boldsymbol{G}^{l-1}$. At the final layer of this two-stream chain, the model outputs the strictly causal representation $\boldsymbol{G}^{L^{2s}}$. By leveraging both causal and strictly causal representations throughout this chain, the model is able to produce a deeply contextualized and expressive strictly causal output. A related two-stream mechanism was previously shown effective in XLNet (Yang et al., 2019), which generalized autoregressive pretraining by unifying both autoencoding-style and autoregressive objectives.

Model Architecture. The final model is composed of two components: (i) an L^{2s}-layer two-stream stack and (ii) a subsequent $(L - L^{2s})$-layer standard causal stack. This composition yields a deep strictly causal model that captures the conditional probability $p_\theta\left(\boldsymbol{x}_n \mid [\boldsymbol{x}_i \mid \mathcal{B}(i) < \mathcal{B}(n)]\right)$, ensuring that each token $\boldsymbol{x}_n$ is generated exclusively based on preceding blocks. Moreover, relative position information is integrated into all attention operations using Rotary Positional Embedding (RoPE) (Su et al., 2024). As noted in recent works (Ou et al., 2024; Zheng et al., 2024), it is unnecessary to explicitly parameterize time-dependence in MDMs. A schematic illustration of the full architecture is shown in Figure 3.

Remark 1. *As discussed above, this design is inherently based on causal attention mechanisms, thereby enabling efficient key-value (KV) caching for fast decoding—a key advantage over BERT-style models commonly used in MDMs. Moreover, since the architecture generalizes autoregressive models (ARMs), it naturally supports hybrid training schemes such as progressive permutation training (further detailed in section 5), allowing the model to learn effectively from both forward (left-to-right) and randomly permuted orderings. Finally, the same design principles suggest that the architecture can be readily adapted to finetune pretrained Large Language Models (LLMs) into diffusion models with simple modifications, presenting a promising direction for future research.*

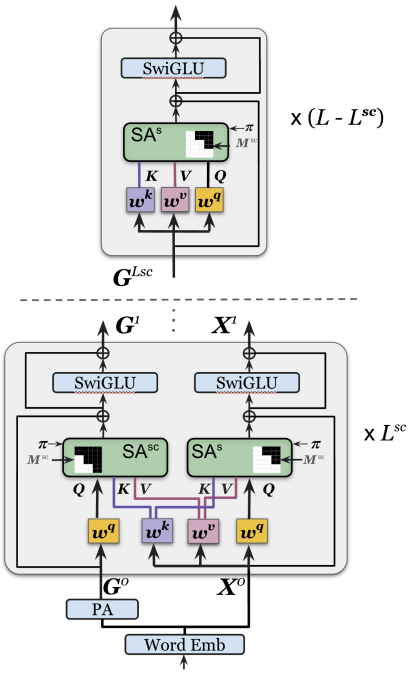

Figure 3: Schematic of the deep strictly causal architecture. The model is composed of L^{2s} two-stream attention layers (causal and strictly causal streams), followed by $L - L^{2s}$ causal layers. The strictly causal stream ($\boldsymbol{G}^l$) captures rich contextual and positional representations using only past block information, while the causal stream ($\boldsymbol{X}^l$) serves as a shared source of key and value features for the attention layers in both streams. The two streams differ in their query input and masking operations. The final output is taken from the top layer, enabling the model to parametrize the conditional distribution $p_\theta(\boldsymbol{x}_n \mid \{\boldsymbol{x}_i \mid \mathcal{B}(i) < \mathcal{B}(n)\})$ while preserving permutation equivariance with respect to the condition set. Layer-Norm, drop-out, and output projections of the attention layers are dropped for simplicity.

3.1 Strided Parallel Generation

To leverage the flexibility of ARMD for efficient generation, we introduce a *strided block-parallel (SBP)* sampling strategy. This method partitions the target sequence of length N into S parallel *streams*, enabling simultaneous token generation while preserving global coherence. Given a parallelism factor S, we first partition the sequence into S equal-sized streams, each of length N/S. The indices are then permuted to interleave these streams, grouping tokens that share the same relative position within their respective streams. For

example: for $N = 8,\ S = 2$: streams: $\mathcal{S}_1 = [1, 2, 3, 4]$, $\mathcal{S}_2 = [5, 6, 7, 8] \rightarrow$ permutation $\pi = [1, 5, 2, 6, 3, 7, 4, 8]$.

The generation process proceeds in two distinct phases: 1) We first generate the initial token (the *head*) of each stream sequentially. In the example above, tokens (1) and (5) are generated autoregressively. This establishes a global context anchor for each stream. 2) *Parallel Block Generation*: Subsequent tokens are generated in parallel groups of size S. Specifically, for each relative position $j > 1$, the model generates the set $\{x_j^{(1)}, x_j^{(2)}, \ldots, x_j^{(S)}\}$ simultaneously. In the example above, the block $(2, 6)$ is generated in parallel, followed by the block $(3, 7)$. Figure 2 illustrates this parallel generation strategy.

This strategy maximizes the physical distance between tokens generated in parallel. For instance, tokens (2) and (6) are separated by N/S positions in the original sequence. This large separation supports the approximate conditional independence assumption required for valid parallel diffusion sampling, i.e., $p(x_2, x_6 \mid \text{context}) \approx p(x_2 \mid \text{context})p(x_6 \mid \text{context})$, while accelerating inference by a factor of S. This strided parallel generation is substantially different from the standard autoregressive block-generation used in BD3-LM (Arriola et al., 2025).

4 RELATED WORKS

Discrete Diffusion Models. A general framework for diffusion models in discrete space, D3PM, was introduced by Austin et al. (2021). This framework is built on a Markov forward process defined by arbitrary transition matrices and optimizes a variational lower bound on the data likelihood. D3PM was initially applied to language modeling at the character level and, while promising in theory, its early applications struggled to match the empirical performance of autoregressive models. More recently, Lou et al. (2023) advanced the field by adapting score matching to discrete domains using a novel score entropy objective. Their model, SEDD, demonstrated strong performance on standard language modeling tasks, narrowing the performance gap with ARMs by achieving competitive perplexity and efficient sampling in discrete diffusion for language. Among the various discrete diffusion strategies developed within these frameworks, masked diffusion models (MDMs) have demonstrated to be a particularly effective approach Austin et al. (2021); Lou et al. (2023), which revealed a conceptual connection to Masked Language Models (MLMs) like BERT (Devlin et al., 2018).

Masked Diffusion Models. Recent works have rapidly advanced MDMs, enhancing their training efficiency and generative quality to achieve performance competitive with ARMs of a similar size. These efforts include exploring simplified objectives and properties of absorbing discrete diffusion models (Shi et al., 2024; Sahoo et al., 2024; Ou et al., 2024). For instance, Sahoo et al. (2024) developed simplified variational objectives while their resulting method supports semi-autoregressive generation that admits efficient generation of sequences of arbitrary length. Building on SEDD, Ou et al. (2024) provided a deeper theoretical understanding of absorbing discrete diffusion. They proposed a reparameterization of absorbing discrete diffusion models to characterize time-independent conditional probabilities of clean data, demonstrating improvements in sampling efficiency and zero-shot perplexity compared to SEDD. The scaling law properties of MDMs on text data have been demonstrated in Nie et al. (2024), which also proposed an unsupervised classifier-free guidance technique (Ho and Salimans, 2022) to further improve inference efficiency on text. Another promising direction for scaling diffusion models involves adapting existing pre-trained AR models to text diffusion models by gradually annealing the attention mask during fine-tuning (Gong et al., 2024), a technique that leverages the connections between AR and diffusion objectives. Recently, Arriola et al. (2025) proposed block diffusion, a hybrid approach that combines discrete diffusion within token blocks and ARMs across blocks, which improves inference efficiency and supports flexible-length sequence generation.

Continuous Diffusion in Latent Space. A separate line of research explores continuous diffusion for discrete data by embedding tokens into continuous latent spaces and applying continuous diffusion on these latent representations (Dieleman et al., 2022; Li et al., 2022; Chen et al., 2022; Gulrajani and Hashimoto, 2023; Han et al., 2022; Lovelace et al., 2023; Graves et al., 2023). However, the process of decoding from continuous back to discrete space remains challenging and is often prone to information loss. Consequently, these latent diffusion models have generally underperformed ARMs on text generation tasks.

Order Agnostic ARMs. Order Agnostic ARMs (OA-ARMs) (Uria et al., 2014) represent a class of generative models that generalizes traditional ARMs by marginalizing over all possible generation orders, thereby addressing the inherent left-to-right bias of ARMs. This generalization allows models to flexibly condition on arbitrary subsets of observed tokens, thereby supporting flexible non-sequential generation. Recent works have established the underlying connections between MDMs and OA-ARMs. For instance, Hoogeboom et al. (2021) showed that absorbing dis-

Table 1: **Zero-shot language modeling perplexity (↓)** : D3PM (Austin et al., 2021), PLAID (Gulrajani and Hashimoto, 2023) and SEDD-Uniform (Lou et al., 2023) are discrete diffusion models, while SEDD (Lou et al., 2023) and RADD (Ou et al., 2024) are masked language models. The number of training steps for each model is presented in parenthesis. *Results for baseline diffusion models are taken from the reported upper bounds in Lou et al. (2023); Ou et al. (2024). The best result for each dataset is shown in **bold**, and the second-best is underlined.

Method	LAMBADA	WikiText2	PTB	WikiText103	1BW
small size models					
GPT-2*	45.04	42.43	138.43	41.60	75.20
D3PM*	93.47	77.28	200.82	75.16	138.92
PLAID (600K)*	57.28	51.80	142.60	50.86	91.12
SEDD-Uniform (400K)*	65.40	50.27	140.12	49.60	101.37
SEDD (400K)*	50.92	41.84	114.24	40.62	79.29
RADD (400K)*	51.70	39.98	**107.85**	37.98	72.99
ARMD (180K)	**44.66**	36.25	130.31	35.66	53.18
ARMD (400K)	45.35	**35.64**	123.43	**35.15**	**52.94**
medium size models					
GPT-2*	**35.66**	31.80	123.14	31.39	55.72
SEDD (400K)*	42.77	31.04	87.12	29.98	61.19
RADD (400K)*	44.10	30.60	**82.08**	29.29	60.32
ARMD (120K)	40.62	27.57	105.09	27.09	45.49
ARMD (300K)	39.08	**26.06**	97.75	**25.55**	**43.91**

Table 2: Test perplexities (ppl; ↓) on LM1B dataset. Baseline models: BERT-mouth (Wang and Cho, 2019), D3PM, DiffusionBert (He et al., 2022), Diffusion-LM (Li et al., 2022), SEDD, MDLM (Sahoo et al., 2024) and BD3-LMs (Arriola et al., 2025). *Results for baseline from: Lou et al. (2023) and BD3-LMs, arriola2025block. The number of training steps is presented in parenthesis. Best diffusion perplexity is shown in **bold**.

Method	ppl (↓)
Autoregressive	
Transformer-XBase*	23.5
Transformer*	22.83
Diffusion	
D3PM-absorb*	82.34
Diffusion-LM*	118.62
DiffusionBert*	63.78
SEDD (1M)*	32.68
MDLM (1M)*	31.78
BD3-LMs L'= 16 (1M)*	30.60
BD3-LMs L'= 4 (1M)*	28.23
ARMD (300K)	23.64
ARMD (1M)	**22.36**

crete diffusion models are equivalent to OA-ARMs under mild assumptions, leading to the proposal of autoregressive diffusion models (ARDMs). Ou et al. (2024) further formally unified the training objectives of absorbing discrete diffusion and OA-ARMs, bridging the practical gap between these two models. From a practical standpoint, since considering all permutations is computationally infeasible, Shih et al. (2022) improve training efficiency and performance by approximating the marginalization through sampling from a small, carefully chosen subset of orderings. In practice, both OA-ARMs and MDMs are trained using a collection of forward passes of masked language models (MLMs)—where a bidirectional model is trained to recover a randomly masked subset of tokens at each call. In contrast, our proposed model enables the parallel evaluation of the entire likelihood within a single forward pass.

5 EXPERIMENTS

We evaluate the proposed model, ARMD, on standard language modeling tasks to demonstrate its effectiveness and training efficiency relative to established generative language models.

Setup. Following the experimental setup of (Lou et al., 2023), we train two versions of ARMD with parameter counts similar to GPT-2 small (125M) and GPT-2 medium (345M) (Radford et al., 2019). As in (Lou et al., 2023; Ou et al., 2024), we train ARMD on the OpenWebText dataset (Gokaslan and Cohen, 2019), a large-scale corpus derived from web pages, widely used for training language models. Each training iteration uses a batch of 512 sequences, each 1024 tokens long, amounting to approximately 0.5 million tokens per batch. For the ARMD architecture, the first half of the transformer layers are configured as the two-stream module, *i.e.* $L^{2s} = L/2$. Further details are provided in Appendix C.

We compare ARMD against several established and recent diffusion models of similar size, including: GPT-2 (Radford et al., 2019), D3PM (Austin et al., 2021), PLAID (Gulrajani and Hashimoto, 2023), SEDD (Lou et al., 2023) and RADD (Ou et al., 2024). Performance is measured using zero-shot perplexity on a suite of widely-used benchmark datasets: LAMBADA (Paperno et al., 2016), Penn Tree Bank (PTB) (Marcus et al., 1993), WikiText2, WikiText103, (Merity et al., 2016), and One Billion Words (1BW) (Chelba et al., 2013).

Progressive Permutation Schedule. The flexible permutation-equivariant design of ARMD enables it to learn from orders beyond the canonical left-to-right. To leverage this, we introduce a *progressive permutation schedule* during training to encourage learning random orderings. This strategy begins by training the model exclusively on forward-ordered sequences for an initial phase ($i_{AR} = 9K$ iterations). Subsequently, we gradually introduce random permutations by progressively

Table 3: **Sample Generation vs. Block Size (Parallelism):** Comparison of ARMD using sequential generation ($T = 1024$) versus parallel generation with $T = 512$ and $T = 256$, corresponding to block sizes of $S = 2$ and $S = 4$, respectively. Performance is reported as Perplexity(↓) + (Entropy). ARMD's average sampling time per sequence is 3.51s for $S = 1$, 1.78s for $S = 2$, and 0.93s for $S = 4$ on one H100 GPU. *Results for the baselines are adopted from Zheng et al. (2024).

Model	T=1024 (Sequential)	T=512 (S=2)	T=256 (S=4)
AR Baseline*	∼36 (8.1)	-	-
SEDD*	∼106 (8.1)	∼110 (8.1)	-
MDLM*	∼103 (8.1)	∼113 (8.1)	-
ARMD (ρ=32, i_{SBP}=10K)	38.7 (8.0)	43.2 (8.0)	47.8 (8.1)
ARMD (ρ=32, i_{SBP}=50K)	39.1 (8.0)	40.1 (8.1)	43.2 (8.1)
ARMD (ρ=32, i_{SBP}=80K)	**36.5** (7.9)	**37.8** (8.0)	**39.4** (8.0)
ARMD (ρ=16, i_{SBP}=10K)	50.0 (8.1)	54.6 (8.1)	64.7 (8.2)
ARMD (ρ=16, i_{SBP}=50K)	45.7 (8.0)	46.1 (8.1)	50.1 (8.1)
ARMD (ρ=16, i_{SBP}=80K)	39.4 (8.0)	40.5 (8.1)	43.7 (8.1)

increasing the number of shuffled tokens from one up to a maximum of $\rho = 32$ by iteration $i_{perm} = 48K$ and remain the same over the rest of training. This curriculum enables the model to become increasingly robust to token permutations while still retaining the left-to-right order as the primary structural prior. Since natural language inherently exhibits a strong left-to-right structure, leveraging this inductive bias as guidance ensures that the model retains strong performance in capturing forward dependencies while learning random permutations.

As shown in Table 1, ARMD achieves state-of-the-art zero-shot performance across most language modeling benchmarks, for both model sizes. Notably, it outperforms most baselines despite being trained for significantly fewer iterations (180K for the small model and 120K for the medium model). This result underscores the superior training efficiency that is a key advantage of our proposed model. We provide extended results, ablation studies, and sample generations in Appendix C.

Parallel Sample Generation. We evaluate the generative perplexity of models in the small parameter setting trained on the OpenWebText dataset. For ARMD, we first train a base model for 400K iterations using the progressive permutation schedule (up to ρ). This is followed by a short fine-tuning phase of i_{SBP} steps (where $i_{\text{SBP}} \in \{10\text{K}, 50\text{K}, 80\text{K}\}$), utilizing the *random strided block permutation* (SBP), where the sequence is partitioned into S equal-sized blocks and then permuted to interleave these blocks, grouping tokens that share the same relative position. During this phase, the block size S is sampled randomly from $\{1, 2, 4\}$ at each step to encourage robust parallel generation.

As the results in Table 3 demonstrate, while standard masked diffusion models (SEDD, MDLM) struggle with high perplexity during parallel generation, ARMD (particularly with $\rho = 32$) efficiently generates high-quality, diverse text (quantified by high mean entropy) with only 10K additional training steps. Notably, the total training budget for ARMD (410K steps) remains lower than that of the AR baseline (500K steps) and standard MDMs (1M steps). Furthermore, increasing i_{SBP} significantly improves the parallel generation and effectively bridges its performance gap with sequential generation.

One Billion Words Benchmark. We also train language models on One Billion Words (LM1B) dataset (Chelba et al., 2013). We follow the training and architecture setting of He et al. (2022); Lou et al. (2023); Sahoo et al. (2024): the model size matches GPT-2 small, and the `bert-base-uncased` tokenizer is applied. ARMD employs two-stream modules in the first half of its layers *i.e.* $L^{2s} = L/2 = 6$. For progressive permutation schedule of ARMD, we begin introducing random permutations at iteration $i_{AR} = 100$K, gradually increasing to a maximum of $\rho = 8$ by iteration $i_{perm} = 300$K.

As shown in Table 2, ARMD achieves the best performance among all diffusion models with significantly fewer training steps. Notably, it outperforms MDMs with 3× fewer training steps, reconfirming the superior training efficiency of our proposed architecture.

6 CONCLUSION

In this work, we introduced ARMD, a novel parallel masked diffusion model, that bridges the gap between diffusion-based learning and autoregressive modeling for language models. By reframing the masked denoising process as a block-wise causal problem, we developed a strictly causal, permutation-equivariant, attention architecture that enables fully parallelized training. This flexible design supports a progressive permutation schedule for learning from both fixed and random token orderings. Importantly, our architecture enables a strided parallel generation strategy that accelerates inference while effectively bridging the performance gap between parallel and sequential generation. Experimental results demonstrate that ARMD achieves state-of-the-art performance on standard language modeling benchmarks, outperforming both ARM and MDM baselines with significantly fewer training iterations. By combining the training efficiency of ARMs with the inherent flexibility of diffusion models, ARMD stands as a scalable and powerful alternative for masked diffusion models.

References

OpenAI. Gpt-4 technical report, 2023.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Hugo Larochelle and Iain Murray. The neural autoregressive distribution estimator. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 29–37. JMLR Workshop and Conference Proceedings, 2011.

Samy Bengio and Yoshua Bengio. Taking on the curse of dimensionality in joint distributions using neural networks. *IEEE Transactions on Neural Networks*, 11(3):550–557, 2000.

Aäron Van Den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In *International conference on machine learning*, pages 1747–1756. PMLR, 2016a.

Aaron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, Koray Kavukcuoglu, et al. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 12, 2016b.

Renjie Liao, Yujia Li, Yang Song, Shenlong Wang, Will Hamilton, David K Duvenaud, Raquel Urtasun, and Richard Zemel. Efficient graph generation with graph recurrent attention networks. *Advances in neural information processing systems*, 32, 2019.

Mahdi Karami. HiGen: Hierarchical graph generative networks. In *The Twelfth International Conference on Learning Representations*, 2024.

Lukas Berglund, Meg Tong, Max Kaufmann, Mikita Balesni, Asa Cooper Stickland, Tomasz Korbak, and Owain Evans. The reversal curse: Llms trained on" a is b" fail to learn" b is a". *arXiv preprint arXiv:2309.12288*, 2023.

Nan Ding, Tomer Levinboim, Jialin Wu, Sebastian Goodman, and Radu Soricut. Causallm is not optimal for in-context learning. *arXiv preprint arXiv:2308.06912*, 2023.

Gregor Bachmann and Vaishnavh Nagarajan. The pitfalls of next-token prediction. *arXiv preprint arXiv:2403.06963*, 2024.

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pages 2256–2265. pmlr, 2015.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in neural information processing systems*, 32, 2019.

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.

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

Ishaan Gulrajani and Tatsunori B Hashimoto. Likelihood-based diffusion language models. *Advances in Neural Information Processing Systems*, 36:16693–16715, 2023.

Jingyang Ou, Shen Nie, Kaiwen Xue, Fengqi Zhu, Jiacheng Sun, Zhenguo Li, and Chongxuan Li. Your absorbing discrete diffusion secretly models the conditional distributions of clean data. *arXiv preprint arXiv:2406.03736*, 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. *arXiv preprint arXiv:2502.09992*, 2025.

Clement Vignac, Igor Krawczuk, Antoine Siraudin, Bohan Wang, Volkan Cevher, and Pascal Frossard. Digress: Discrete denoising diffusion for graph generation. *arXiv preprint arXiv:2209.14734*, 2022.

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.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning*, pages 5156–5165. PMLR, 2020.

Weizhe Hua, Zihang Dai, Hanxiao Liu, and Quoc Le. Transformer quality in linear time. In *International conference on machine learning*, pages 9099–9117. PMLR, 2022.

Praneeth Kacham, Vahab Mirrokni, and Peilin Zhong. Polysketchformer: Fast transformers via sketching polynomial kernels. In *Forty-first International Conference on Machine Learning*, 2024. URL `https://openreview.net/forum?id=ghYrfdJfjK`.

Guy E Blelloch. Prefix sums and their applications. 1990.

Jimmy T.H. Smith, Andrew Warrington, and Scott Linderman. Simplified state space layers for sequence modeling. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=Ai8Hw3AXqks.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32, 2019.

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

Marianne Arriola, Aaron Gokaslan, Justin T Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Block diffusion: Interpolating between autoregressive and diffusion language models. *arXiv preprint arXiv:2503.09573*, 2025.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

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.

Shen Nie, Fengqi Zhu, Chao Du, Tianyu Pang, Qian Liu, Guangtao Zeng, Min Lin, and Chongxuan Li. Scaling up masked diffusion models on text. *arXiv preprint arXiv:2410.18514*, 2024.

Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022.

Shansan Gong, Shivam Agarwal, Yizhe Zhang, Jiacheng Ye, Lin Zheng, Mukai Li, Chenxin An, Peilin Zhao, Wei Bi, Jiawei Han, et al. Scaling diffusion language models via adaptation from autoregressive models. *arXiv preprint arXiv:2410.17891*, 2024.

Sander Dieleman, Laurent Sartran, Arman Roshannai, Nikolay Savinov, Yaroslav Ganin, Pierre H Richemond, Arnaud Doucet, Robin Strudel, Chris Dyer, Conor Durkan, et al. Continuous diffusion for categorical data. *arXiv preprint arXiv:2211.15089*, 2022.

Xiang Li, John Thickstun, Ishaan Gulrajani, Percy S Liang, and Tatsunori B Hashimoto. Diffusion-lm improves controllable text generation. *Advances in neural information processing systems*, 35:4328–4343, 2022.

Ting Chen, Ruixiang Zhang, and Geoffrey Hinton. Analog bits: Generating discrete data using diffusion models with self-conditioning. *arXiv preprint arXiv:2208.04202*, 2022.

Xiaochuang Han, Sachin Kumar, and Yulia Tsvetkov. Ssd-lm: Semi-autoregressive simplex-based diffusion language model for text generation and modular control. *arXiv preprint arXiv:2210.17432*, 2022.

Justin Lovelace, Varsha Kishore, Chao Wan, Eliot Shekhtman, and Kilian Q Weinberger. Latent diffusion for language generation. *Advances in Neural Information Processing Systems*, 36:56998–57025, 2023.

Alex Graves, Rupesh Kumar Srivastava, Timothy Atkinson, and Faustino Gomez. Bayesian flow networks. *arXiv preprint arXiv:2308.07037*, 2023.

Alex Wang and Kyunghyun Cho. Bert has a mouth, and it must speak: Bert as a markov random field language model. *arXiv preprint arXiv:1902.04094*, 2019.

Zhengfu He, Tianxiang Sun, Kuanning Wang, Xuanjing Huang, and Xipeng Qiu. Diffusionbert: Improving generative masked language models with diffusion models. *arXiv preprint arXiv:2211.15029*, 2022.

Benigno Uria, Iain Murray, and Hugo Larochelle. A deep and tractable density estimator. In *International Conference on Machine Learning*, pages 467–475. PMLR, 2014.

Emiel Hoogeboom, Alexey A Gritsenko, Jasmijn Bastings, Ben Poole, Rianne van den Berg, and Tim Salimans. Autoregressive diffusion models. *arXiv preprint arXiv:2110.02037*, 2021.

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.

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

Aaron Gokaslan and Vanya Cohen. Openwebtext corpus. http://Skylion007.github.io/OpenWebTextCorpus, 2019.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Quan Ngoc Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The lambada dataset: Word prediction

requiring a broad discourse context. *arXiv preprint arXiv:1606.06031*, 2016.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. *Comput. Linguistics*, 19:313–330, 1993.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *International Conference on Learning Representations*, 2016.

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, T. Brants, Phillip Todd Koehn, and Tony Robinson. One billion word benchmark for measuring progress in statistical language modeling. In *Interspeech*, 2013.

Subham Sekhar Sahoo, Justin Deschenaux, Aaron Gokaslan, Guanghan Wang, Justin Chiu, and Volodymyr Kuleshov. The diffusion duality. *arXiv preprint arXiv:2506.10892*, 2025.

Zhanqiu Hu, Jian Meng, Yash Akhauri, Mohamed S Abdelfattah, Jae-sun Seo, Zhiru Zhang, and Udit Gupta. Accelerating diffusion language model inference via efficient kv caching and guided diffusion. *arXiv preprint arXiv:2505.21467*, 2025.

Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Dream 7b, 2025. URL `https://hkunlp.github.io/blog/2025/dream`.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes]
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Yes]
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [Not Applicable]

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [Yes]
   (b) Complete proofs of all theoretical results. [Yes]
   (c) Clear explanations of any assumptions. [Yes]

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [Yes]
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Yes]
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Not Applicable]
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. [Yes]
   (b) The license information of the assets, if applicable. [Not Applicable]
   (c) New assets either in the supplemental material or as a URL, if applicable. [Not Applicable]
   (d) Information about consent from data providers/curators. [Not Applicable]
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable]

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. [Not Applicable]
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]

A NOTATION DEFINITION

Notations	Brief definition and interpretation
N	Sequence length (number of tokens).
d	Dimensionality of token embeddings.
T	Number of diffusion steps (masking timesteps).
K	Number of categories (vocabulary size for discrete tokens).
$\boldsymbol{z}_j = \boldsymbol{z}_j^0$	Clean sample at position j, represented as a one-hot vector $\boldsymbol{z}_j^0 \in \{0,1\}^K$.
$\boldsymbol{z}_j^t$	Noisy (partially masked) version of token j at timestep t in the diffusion process.
$\boldsymbol{z}_{1:N}$	Original input sequence before masking.
$\tau(j)$	Masking timestep when token j is first masked during the forward diffusion process.
$\pi,\ \pi(\cdot)$	$i = \pi(j)$, Permutation that sorts token indices by $\tau(j)$ in descending order. Also, as a function it permutes a sequence along the sequence dimension.
$\hat{\pi}(\cdot)$	Inverse permutation: $j = \hat{\pi}(i) = \pi^{-1}(i)$.
$\mathcal{U}(\cdot)$	Uniform distribution
Π_N	Set of all permutations of the indices $\{1, ..., N\}$
$\boldsymbol{x}_{1:N} = \pi(\boldsymbol{z}_{1:N})$	Reordered sequence where tokens masked last come first. $\boldsymbol{x}_{1:N} = (\boldsymbol{z}_{\pi(1)}, \boldsymbol{z}_{\pi(2)}, ..., \boldsymbol{z}_{\pi(N)})$
$\boldsymbol{X} = [\mathcal{X}^{(1)}, \ldots, \mathcal{X}^{(T)}]$	Partition of $\boldsymbol{x}_{1:N}$ into T non-overlapping blocks by unmasking order.
$\mathcal{X}^{(t)}$	Block of tokens generated at step t (with size $n_t \times d$), where n_t is number of tokens in $\mathcal{X}^{(t)}$ with $\sum_{t=1}^{T} n_t = N$.
$\mathcal{B}(n)$	Block index that token $\boldsymbol{x}_n$ belongs to. $\mathcal{B}(n) = T - \tau(n) + 1$.
$\gamma(t)$	Reweighting factor for loss at step t (from diffusion schedule).
$\mathbf{1}$	All-ones vector.
Δ^{K-1}	The K-simplex: set of all K-dimensional probability vectors.
$\boldsymbol{\phi} \in \Delta^{K-1}$	Limit (stationary) distribution of the diffusion process
$\boldsymbol{m} \in \{0,1\}^{K+1}$	One-hot representation over K categories + 1 special mask token.
$\mathrm{Cat}(\boldsymbol{z};\ \boldsymbol{\eta})$	Categorical distribution over $\boldsymbol{z}$ with probabilities of the classes $\boldsymbol{\eta}$.
$q(\boldsymbol{z}^t \mid \boldsymbol{z}^{t-1})$	Forward diffusion (masking) transition probability.
$p_\theta(\ \mid\ )$	Learned reverse (unmasking) model parameterized by θ.
$D_{\mathrm{KL}}(q \,\|\, p)$	Kullback–Leibler divergence between distributions q and p.
$\boldsymbol{x}_{<n}$	Subsequence of tokens before token n in a predefined order.
$f^c()$	causal function: $y_n = f^c(x_{\leq n})$
$f^{sc}()$	strictly causal function: $y_n = f^{sc}(x_{<n})$
$\boldsymbol{X}$	Input matrix composed of token embeddings: $\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N]^\top \in \mathbb{R}^{N \times d}$
$\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$	Query, key, and value matrices from linear projections of $\boldsymbol{X}$.
$\hat{\boldsymbol{q}}_n$	Strictly causal query at position n, independent of block $\mathcal{B}(n)$
$\boldsymbol{W}^q, \boldsymbol{W}^k, \boldsymbol{W}^v$	Linear projection matrices for computing $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$.
$\boldsymbol{X}^l$	Causal hidden representation at layer l
$\boldsymbol{G}^l$	Strictly causal hidden representation at layer l
$\mathrm{SA}^c(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}; \boldsymbol{M}^c)$	Causal self-attention layer: $\mathrm{Softmax}(\boldsymbol{Q}\boldsymbol{K}^\top + \boldsymbol{M}^c)\,\boldsymbol{V}$.
$\mathrm{SA}^{sc}(\hat{\boldsymbol{Q}}, \boldsymbol{K}, \boldsymbol{V}; \boldsymbol{M}^{sc})$	Strictly causal self-attention layer: $\mathrm{Softmax}(\hat{\boldsymbol{Q}}\boldsymbol{K}^\top + \boldsymbol{M}^{sc})\,\boldsymbol{V}$.
$\boldsymbol{M}^c, \boldsymbol{M}^{sc}$	Causal and strictly attention masks: $M^{sc}_{(n,i)} = \begin{cases} 0 & \text{if } \mathcal{B}(i) < \mathcal{B}(n) \\ -\infty & \text{otherwise} \end{cases}$, $M^{c}_{(n,i)} = \begin{cases} 0 & \text{if } \mathcal{B}(i) \leq \mathcal{B}(n) \\ -\infty & \text{otherwise} \end{cases}$
f_q^{sc}	Strictly causal function used to compute $\hat{\boldsymbol{q}}_n$
$\mathrm{PA}^{sc}(\cdot) = \mathrm{FF}^{sc}(\cdot)$	Prefix aggregation layer: a strictly causal feedforward layer for aggregating past information.
h_{ni}	Weight of $\mathrm{PA}^{sc}(\cdot)$ from position i to position n
$\mathbf{PE},\ \mathbf{PE}_j,\ \hat{\mathbf{PE}}$	$\mathbf{PE}$: positional embeddings; $\mathbf{PE}_j$: embedding for position j; $\hat{\mathbf{PE}}$: permuted embeddings using π.
$p_\theta(\boldsymbol{x}_n \mid [\boldsymbol{x}_i \mid \mathcal{B}(i) < \mathcal{B}(n)])$	Autoregressive conditional probability of $\boldsymbol{x}_n$ given strictly causal context in MDMs.

B EXTENDED DISCUSSION

B.1 Order Agnostic ARM

Order Agnostic ARMs (OA-ARMs) (Uria et al., 2014) compute the data likelihood by marginalizing over all possible generation orders. This approach enables the model to learn conditional generative distributions over any order, thereby mitigating the left-to-right bias inherent in traditional ARMs, and also allowing the model to flexibly condition on arbitrary subsets of observed tokens, supporting non-sequential generation. The OA-ARM

training objective can be reformulated as (Hoogeboom et al., 2021; Shih et al., 2022):

$$\log p(\boldsymbol{z}_{1:N}) \geq \mathbb{E}_{\pi\sim\mathcal{U}(\Pi_N)} \sum_{t=1}^{N} \underbrace{\log p(\boldsymbol{z}_{\pi(t)} \mid \boldsymbol{z}_{\pi(<t)})}_{L_t} \tag{8}$$

$$= \mathbb{E}_{\pi\sim\mathcal{U}(\Pi_N)} N \cdot \mathbb{E}_{t\sim\mathcal{U}(\{1,\ldots,N\})} \log p(\boldsymbol{z}_{\pi(t)} \mid \boldsymbol{z}_{\pi(<t)})$$

$$= N \cdot \mathbb{E}_{t\sim\mathcal{U}(\{1,\ldots,N\})} \mathbb{E}_{\pi\sim\mathcal{U}(\Pi_N)} \underbrace{\frac{1}{N-t+1} \sum_{k\in\pi(\geq t)} \log p(\boldsymbol{z}_k \mid \boldsymbol{z}_{\pi(<t)})}_{\tilde{L}_t} \tag{9}$$

The likelihood upper-bound (9) highlights the connection between OA-ARMs and masked diffusion models. Recent work by Ou et al. (2024) formally unifies the training objectives of MDMs and OA-ARMs. However, training OA-ARMs in (Hoogeboom et al., 2021; Shih et al., 2022; Ou et al., 2024) involves optimizing a single timestep objective $\tilde{L}_t$ at a time, similar to masked language models like BERT—by masking random subsets of input tokens and training bidirectional models to recover them. Hence they require longer training than ARMs.

In contrast, we adopt the autoregressive-style likelihood (8) and introduce a strictly causal, permutation-equivariant model that allows parallel evaluation of all conditionals during training. This is achieved through a carefully designed attention masking, described in detail in §3.

Recent Diffusion Language Models. Several recent works have focused on improving the efficiency and scale of Diffusion Language Models (DLMs). Sahoo et al. (2025) propose a curriculum learning strategy guided by the Gaussian process, and aslo present Discrete Consistency Distillation, which adapts consistency distillation from the continuous to the discrete setting. Together, these techniques significantly enhance both training and sampling efficiency. Hu et al. (2025) propose a key-value (KV) approximation caching technique and also employs a lightweight, pretrained autoregressive model to guide the token unmasking process. This method improves the efficiency of DLMs by significantly reducing the total number of required denoising iterations. Recent efforts have focused on scaling DLMs to sizes comparable to autoregressive models. Initial works like DiffuLLaMA (Gong et al., 2024), LLaDA (Nie et al., 2025) successfully scaled DLMs into the billion-parameter range. The recent Dream model (Ye et al., 2025) pushed this frontier further. It scales up a DLM to 7 billion parameters by using training strategies, such as initialization from a pretrained autoregressive LLM and a context-adaptive, token-level noise rescheduling scheme. The resulting model demonstrates superior planning abilities and inference flexibility.

Advantages of ARMD and its Limitations. As discussed in §3, the architecture of ARMD is built on causal attention, enabling efficient decoding and sampling via *key-value (KV) caching*. Since it generalizes autoregressive models (ARMs), ARMD naturally supports hybrid training strategies—such as *progressive permutation training* (section 5) allowing the model to learn from both forward (left-to-right) and random orderings. Its flexible design also makes it amenable to *fine-tuning pretrained Large Language Models (LLMs) into diffusion models* with minimal architectural changes, opening promising directions for future work. In addition, the model's permutation-equivariant property and any-order training allow for a range of sampling strategies, as discussed in §C.4.

On the other hand, the proposed two-stream attention architecture (Yang et al., 2019), which computes both deep causal and strictly causal representations, introduces additional computational overhead, despite sharing parameters and key/value projections and being parallelizable. To mitigate this overhead in deep models, ARMD architecture is configured to use a portion of early layers as two-stream attention rather than all layers, which is studied empirically in the following section.

Proof. ***The Composition of Causal and Strictly Causal Layers:*** We want to show that composing a strictly causal layer with any number of causal ones yields a strictly causal network. We first prove that the composition of a strictly causal layer and a causal layer results in a network that is, overall, strictly causal. A *strictly causal layer*, f^{sc}, where the output at position n depends only on inputs before n. Let $g = f^{sc}(x)$, so the output at position n is $g_n = f^{sc}(x_{<n})$. A *causal layer*, f^c, where the output at position n depends on inputs up to and including n. Let $y = f^c(g)$, so the output at position n is $y_n = f^c(g_{\leq n})$. The composite function is $y = f^c \circ f^{sc} = f^c(f^{sc}(x))$. To prove that this composition is strictly causal, we must show that the final output y_n depends only on the original inputs x at positions less than n (i.e., $x_{<n}$).

The output of the causal layer at position n, y_n, is a function of inputs $\{g_1, g_2, \ldots, g_n\}$. Each of these intermediate outputs, g_i, is an output from the strictly causal layer f^{sc}:

- g_1 depends on $x_{<1}$ (an empty set of inputs).
- g_2 depends on $x_{<2}$ (i.e., just x_1)
- ....
- g_n depends on $x_{<n}$ (i.e., the set $\{x_1, x_2, \ldots, x_{n-1}\}$).

To find the total dependency of y_n, we need to find the set of all original inputs x that are required to compute the set $\{g_1, g_2, \ldots, g_n\}$. This is the union of the dependencies of each individual g_i:

$$\text{Dependency}(y_n) = \text{Dependency}(g_1) \cup \text{Dependency}(g_2) \cup \ldots \cup \text{Dependency}(g_n) = \{x_{<1}\} \cup \{x_{<2}\} \cup \cdots \cup \{x_{<n}\}$$

The union of this collection of sets is simply the largest set in the collection, which is $\{x_{<n}\}$. Therefore, the final output y_n depends only on the inputs strictly preceding it ($x_{<n}$). Hence, the composite function $f^c \circ f^{sc}$ is strictly causal. Also, by induction, the composition of a strictly causal layer with any number of causal ones yields a strictly causal network. □

B.2 Complexity and Latency Analysis

In this section, we analyze the theoretical computational cost and empirical latency of the ARMD architecture.

Theoretical Complexity Analysis. The ARMD architecture consists of two distinct components: 1) A *Two-Stream* attention mechanism for the first L^{2s} layers, which enables the computation of both a Causal stream ($\boldsymbol{X}$) and a Strictly Causal stream ($\boldsymbol{G}$) to guarantee strict causality and permutation equivariance. 2) A *Standard Causal Stack* for the remaining $L - L^{2s}$ layers. Since the attention weights ($\boldsymbol{W}^q, \boldsymbol{W}^k, \boldsymbol{W}^v, \boldsymbol{W}^o$) are shared between streams, the total parameter count remains nearly identical to that of a standard model. We compare the Floating Point Operations (FLOPs) of an ARMD layer against a standard AR transformer layer. Let N be the sequence length, d be the hidden dimension, and assume a SwiGLU Feed-Forward Network (FFN) with an expansion factor of 8/3 (widely used in modern LLMs like Llama-3).

- **Standard Causal Layer (Single Stream):** The Attention Projections ($\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}, \boldsymbol{O}$) require $8Nd^2$ with $4N^2d$ for the attention mechanism, and the FFN (SwiGLU) needs $16Nd^2$ FLOPs. This brings the total cost per layer to:
$$C_{std} = 24Nd^2 + 4N^2d$$
- **Two-Stream Layer:** This layer computes two hidden states ($\boldsymbol{X}$ and $\boldsymbol{G}$). While sharing Key/Value states avoids redundant computation, the strictly causal stream $\boldsymbol{G}$ requires computing its own specific query ($\hat{\boldsymbol{Q}}$) and output ($\boldsymbol{O}$) representations ($4Nd^2$), an independent FFN ($16Nd^2$), and the attention scores ($4N^2d$). This results in the following additional FLOPs for the Strictly Causal stream:
$$C_{extra} = 20Nd^2 + 4N^2d$$
Note: For the final layer of the Two-Stream stack, we only need to update $\boldsymbol{G}$. The computation saved by skipping the update for $\boldsymbol{X}$ is approximately equivalent to the additional computation required for the *Prefix Aggregation* layer ($4Nd^2 + 4N^2d$).

For our default configuration, where the Two-Stream mechanism is applied to the first half of the model ($L^{2s} = L/2$), the total FLOPs ratio relative to a standard AR model is:

$$\text{Ratio} = \frac{\frac{L}{2}(C_{std} + C_{extra}) + \frac{L}{2}C_{std}}{L \cdot C_{std}}$$

Consequently, the architectural overhead is approximately 42% for Dense Operations (d^2 term) and 50% for Attention Operations (N^2 term).

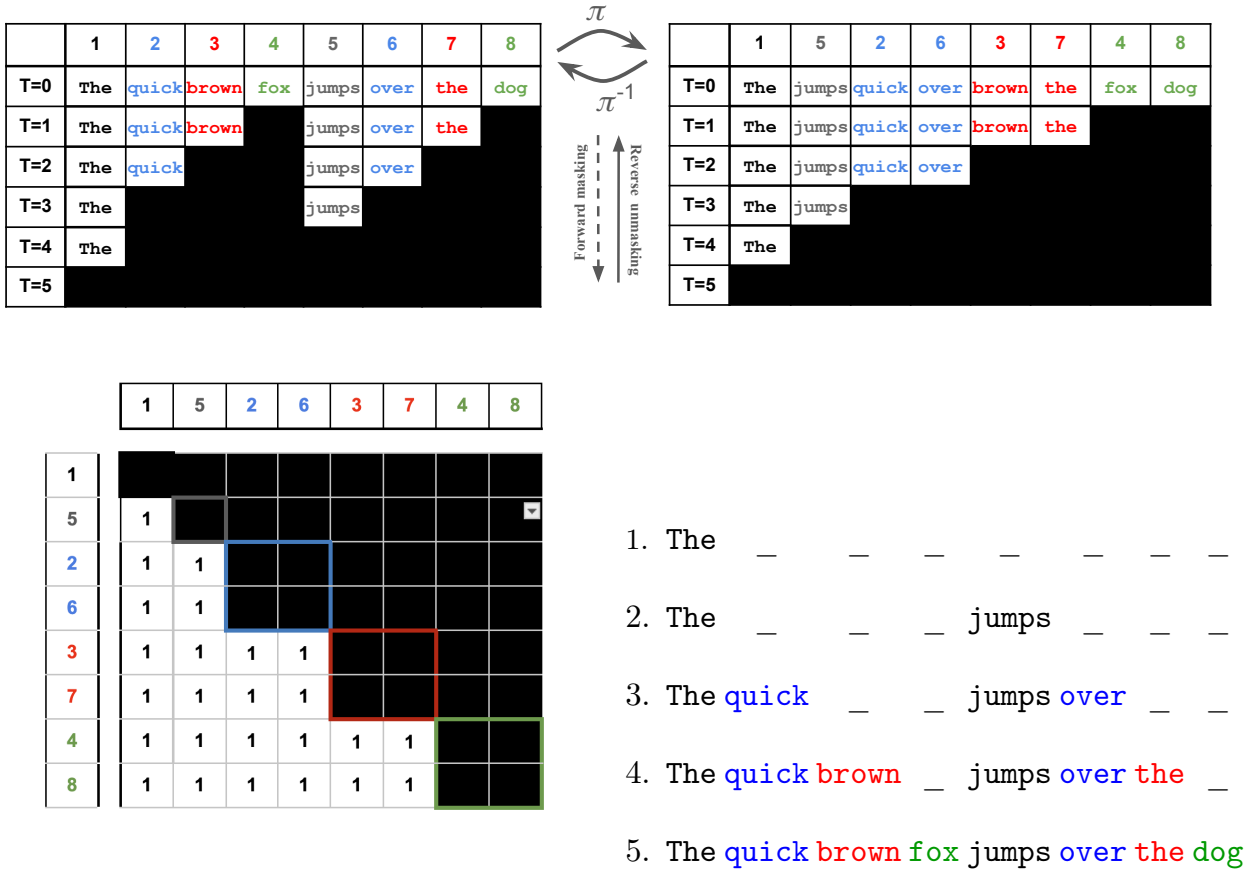

Figure 4: Illustration of the *strided parallel generation*, its diffusion process and masking. *(Top Left)*: An instance of masked diffusion process. *(Top Right)*: The causal patterns by permuting the sequence $\boldsymbol{x}_{1:N} = \pi(\boldsymbol{z}_{1:N})$. *(Bottom left)*: The resulting strictly causal attention mask. *(Bottom right)*: The *strided parallel generation* where steps 1-2 represent the sequential generation of the stream heads, and steps 3-5 show the parallel generation of subsequent tokens. Here, same-colored tokens are generated simultaneously.

Empirical Latency. We benchmarked the wall-clock training throughput (sequences/second) for a 12-layer model on 4× NVIDIA H100 GPUs with a sequence length of 1024. Table 4 summarizes the results.

For our default configuration ($L^{2s} = 6$), the training latency is 1.22× that of a standard AR model. However, this overhead is negligible compared to the 3×–8× reduction in total training steps required for convergence relative to baseline masked diffusion models, resulting in a significantly lower *total training FLOPs* footprint. Furthermore, notably, the two streams within a block operate independently and could be sharded via model parallelism to further close this latency gap; however, the results reported here were measured without such optimizations.

C EXTENDED EXPERIMENTAL RESULTS, DETAILS, AND ABLATIONS

C.1 OpenWebText dataset.

We trained ARMD, on the the OpenWebText dataset (Gokaslan and Cohen, 2019) in two sizes—small and medium—matching the parameter counts of GPT-2 small (125M) and medium (345M) (Radford et al., 2019): small models use $L = 12$ layers with a hidden dimension of $d = 768$ and 12 attention heads while medium size models are composed of $L = 24$ layers with a hidden size of $d = 1024$ and 16 attention heads. The ARMD

Table 4: Training throughput comparison. ARMD incurs a moderate overhead compared to standard AR models, which scales with the number of Two-Stream layers (L^{2s}).

Model Configuration	Throughput (seq/sec)	Relative Overhead
Standard AR	448.02	1.00×
ARMD ($L^{2s} = 3$)	406.35	1.10×
ARMD ($L^{2s} = 6$) *(Default)*	368.53	**1.22**×
ARMD ($L^{2s} = 9$)	338.89	1.32×
ARMD ($L^{2s} = 12$)	312.98	1.43×

architecture is composed of a stack of L^{2s} two-stream layers, followed by $L - L^{2s}$ standard causal layers, as illustrated in Figure 3. For our primary models, we configure the first half of the transformer layers as the two-stream module, i.e. $L^{2s} = L/2$. To study the impact of this choice, we also trained ablation models with different configurations, such as a small model with all layers as two-stream $L^{2s} = 12$ and a medium model with only 1/4 layers as two-stream stack $L^{2s} = 6$. For the prefix aggregation layer (see equation 6), we use sinusoidal positional embeddings (Vaswani et al., 2017) projected through a two-layer MLP with a latent dimension of d/4. Additionally, all Transformer layers incorporate Rotary Positional Embeddings (RoPE) (Su et al., 2024) for relative position awareness.

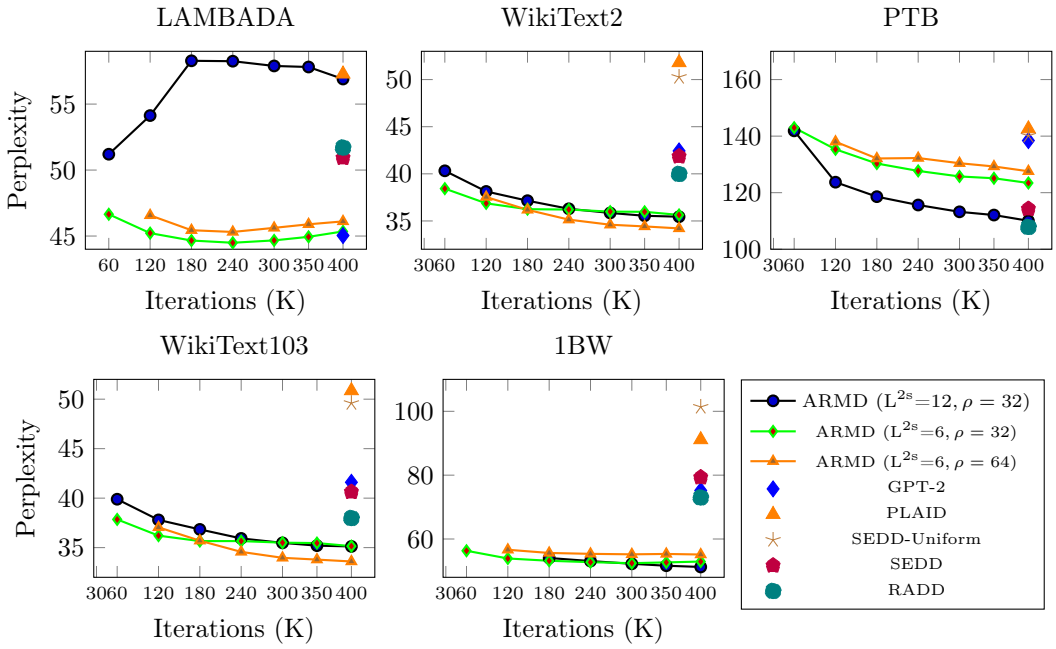

Figure 5: Perplexity vs. training steps for *small-size* models.

Time-Independent Generative Network. Recent studies have indicated that explicit time conditioning in discrete diffusion generative models is either unnecessary or has minimal impact on performance (Ou et al., 2024; Zheng et al., 2024; Sahoo et al., 2024). Consequently, we parameterize our ARMD model with a time-independent neural network, such that $p_\theta(\boldsymbol{x}_n \mid [\boldsymbol{x}_i \mid \mathcal{B}(i) < \mathcal{B}(i)]) = \mathrm{NN}^{sc}([\boldsymbol{x}_i \mid \mathcal{B}(i) < \mathcal{B}(n)])[n]$. We trained ARMD with equal-length masking blocks of size 1, which corresponds to $T = 1024$ diffusion steps during training, and set reweighting factor in the objective equation 2 to constant value, $\gamma_t = 1/T$.

We follow the **training setup** in Lou et al. (2023), using the OpenWebText dataset (Gokaslan and Cohen, 2019). All models are trained for 400K iterations with a batch size of 512 sequences, each 1024 tokens long, amounting to approximately 0.5 million tokens per batch. Tokenization is performed using the `GPT2` tokenizer.

For **optimizer**, we adopted the AdamW optimizer (Loshchilov and Hutter, 2017) with a learning rate of 3×10^{-4}

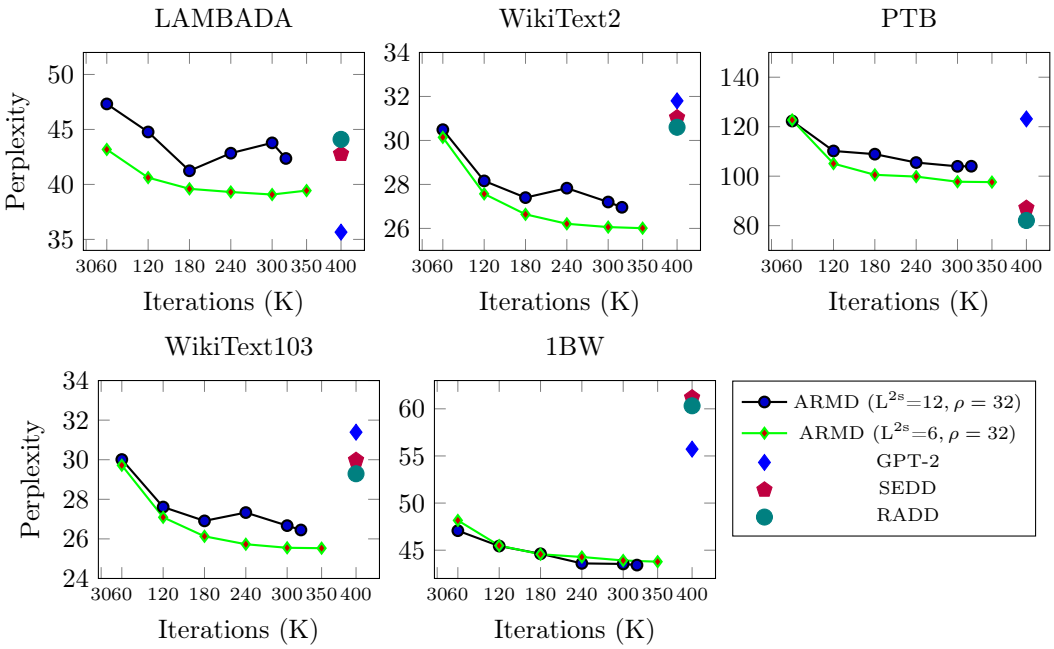

Figure 6: Perplexity vs. training steps for *medium-size* models.

and default hyperparameters ($\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 1 \times 10^{-8}$), and linear warmup over the first 2000 iterations. We apply gradient clipping with a norm threshold of 1, a weight decay of 0.03, and a dropout rate of 0.02. All models were trained on 8× (or 4×) NVIDIA H100 GPUs. Training ARMD-small for 100K iterations took approximately 19 hours, while ARMD-medium required around 50 hours for 100K iterations. For inference and sampling, we used NVIDIA V100 GPUs.

We used a *progressive permutation schedule* during training, where the model is initially trained exclusively on forward-ordered sequences for the first i_{AR} iterations. After this phase, we gradually introduce partial permutations by randomly selecting a small subset of tokens—starting with a single token—and permuting their positions within the sequence. The number of permuted tokens is progressively increased up to a maximum of ρ by iteration i_{perm}. This curriculum enables the model to become increasingly robust to token permutations while still retaining the left-to-right order as the primary structural prior. Since natural language inherently exhibits a strong left-to-right structure, leveraging this inductive bias as guidance ensures that the model retains strong performance in capturing forward dependencies.

We compare ARMD against several established and recent generative models with similar parameter counts, including: GPT-2 (Radford et al., 2019), D3PM (Austin et al., 2021), PLAID (Gulrajani and Hashimoto, 2023), SEDD (Lou et al., 2023) and RADD (Ou et al., 2024). To evaluate language modeling performance of ARMD, we report zero-shot perplexity on widely-used benchmarks: LAMBADA (Paperno et al., 2016), Penn Tree Bank (PTB) (Marcus et al., 1993), WikiText2, WikiText103, (Merity et al., 2016), and One Billion Words (1BW) (Chelba et al., 2013).

As the results in Table 1 and Figure 5, Figure 6 demonstrate, small (medium) ARMD achieves state-of-the-art performance on four (three) benchmarks, while ARMD-small outperforms GPT-2 small across all benchmarks. Notably, it outperforms diffusion baselines in most benchmarks with significantly less training iterations, highlighting the training efficiency that is a key advantage of our model.

C.2 Ablation Studies

We conduct ablation studies for language models trained on OpenWebText dataset to evaluate the effects of the two-stream attention layers and the progressive permutation schedule. Specifically, we train ARMD-small under the following configurations:

1. Small ARMD with $L^{2s} = 12$: all layers use two-stream attention, while the remaining layers use standard

Table 5: **Zero-shot language modeling perplexity (↓)** : D3PM (Austin et al., 2021), PLAID (Gulrajani and Hashimoto, 2023) and SEDD-Uniform (Lou et al., 2023) are discrete diffusion models, while SEDD (Lou et al., 2023) and RADD (Ou et al., 2024) are masked language models. The number of training steps for each model is presented in parenthesis. *Results for baseline diffusion models are taken from the reported upper bounds in Lou et al. (2023); Ou et al. (2024). The best result for each dataset is shown in **bold**, and the second-best is underlined.

Method	LAMBADA	WikiText2	PTB	WikiText103	1BW
small size models					
GPT-2*	<u>45.04</u>	42.43	138.43	41.60	75.20
D3PM*	93.47	77.28	200.82	75.16	138.92
PLAID (600K)*	57.28	51.80	142.60	50.86	91.12
SEDD-Uniform (400K)*	65.40	50.27	140.12	49.60	101.37
SEDD (400K)*	50.92	41.84	<u>114.24</u>	40.62	79.29
RADD (400K)*	51.70	39.98	**107.85**	37.98	72.99
$\{L^{2s}=6,\ \rho=32\}$					
ARMD (60K)	46.65	38.43	142.98	37.85	56.32
ARMD (180K)	**44.66**	<u>36.25</u>	130.31	<u>35.66</u>	<u>53.18</u>
ARMD (400K)	45.35	<u>35.64</u>	123.43	<u>35.15</u>	<u>52.94</u>
$\{L^{2s}=6,\ \rho=64\}$					
ARMD (60K)	47.16	37.83	152.75	37.34	57.26
ARMD (180K)	45.44	36.18	132.10	35.70	55.63
ARMD (400K)	46.11	**34.21**	127.58	**33.60**	55.14
$\{L^{2s}=12,\ \rho=32\}$					
ARMD (180K)	58.28	37.15	118.59	36.84	54.05
ARMD (400K)	56.90	35.45	110.07	35.12	**51.24**
medium size models					
GPT-2*	**35.66**	31.80	123.14	31.39	55.72
SEDD (400K)*	42.77	31.04	87.12	29.98	61.19
RADD (400K)*	44.10	30.60	**82.08**	29.29	60.32
$\{L^{2s}=12,\ \rho=32\}$					
ARMD (60K)	43.18	30.14	122.71	29.72	48.16
ARMD (120K)	40.62	<u>27.57</u>	105.09	<u>27.09</u>	<u>45.49</u>
ARMD (300K)	<u>39.08</u>	**26.06**	97.75	**25.55**	<u>43.91</u>
$\{L^{2s}=6,\ \rho=32\}$					
ARMD (60K)	47.32	30.49	122.29	30.01	47.07
ARMD (120K)	44.77	28.16	110.19	27.61	45.44
ARMD (300K)	43.78	27.20	103.98	26.67	**43.54**

causal attention.

2. Medium ARMD with $L^{2s} = 6$: a first quarter of layers use two-stream attention.
3. Permutation schedule with *32 permutes*: The number of permuted tokens gradually increased, starting at iteration i_{AR}= 9K and reaching the maximum ρ=32 by iteration i_{perm}=48K.
4. Permutation schedule with *64 permutes*: $\rho = 32$, with $i_{AR} = 40$K and $i_{perm} = 120$K.

As shown in Table 5, setting $L^{2s} = L/2$ is enough for achieving strong performance while reducing the number of two-stream layers to $L^{2s} = L/4$ slightly compromises performance—though still comparable and better than the baseline methods—while offering improved computational efficiency. Increasing the number of permutation steps from 32 to 64 further improves performance on some datasets.

C.3 One Billion Words dataset.

We also train language models on One Billion Words (LM1B) dataset (Chelba et al., 2013). We follow the training and architecture setting of He et al. (2022); Lou et al. (2023); Sahoo et al. (2024): the models' size match GPT-2 small (12 layers with a hidden dimension of 768 and 12 attention heads), `bert-base-uncased` tokenizer is applied. ARMD uses two-stream modules for its first half of the layers, *i.e.* $L^{2s} = L/2 = 6$. For training, the sequences are padded and truncated to a length of 128 with batch size of 512. Models are trained for 32B tokens which correspond to 1M steps in diffusion. ($i_{AR} = 9K$ iterations). For progressive permutation schedule of ARMD, starting from iteration $i_{AR} = 100K$, we gradually introduce random permutations by increasing the number of shuffled tokens up to a maximum of $\rho = 8$ by iteration $i_{perm} = 300K$.

For **optimizer**, similar to training for OpenWebText, we use the AdamW optimizer with a learning rate of 3×10^{-4}, a weight decay of 0.03, a linear warmup schedule over the first 2000 iterations with gradient clipping at 1. We use a dropout rate of 0.02.

C.4 Sample Generation

A recent study by Zheng et al. (2024) found that numerical errors in `float32` precision during Gumbel-based categorical sampling—commonly used in MDMs—can lead to deceptively low sample perplexity while producing low token diversity (i.e., low sample entropy). This effectively mimics the effect of using a lower sampling temperature. To mitigate this effect, they suggest using higher precision (`float64`) during sampling. Therefore, we employ `float64` precision for sampling across all MDMs in our experiments. Additionally, we increase the sampling temperature until the generated sequences achieve a level of token entropy comparable to that of MDMs, ensuring diversity in the outputs. Unigram entropy is computed for each generated sequence by measuring the entropy over its token distribution, and then averaged across all sampled sequences.

The performance of sequential and parallel sample generation of the ARMD compared to baseline diffusion models are presented in Table 3, and some examples of the generated texts are presented in the following section.

Sample 1 for model: ARMD-small and sequential generation ($T = 1024$):

keys to functionality. The little messages they're communicating need to be called to send messages, and then they need to have their counterparts in the star system also able to use them. Otherwise, they can't receive messages and lose their abilities. That solution, on the other hand they needed to do something, too – name it "coming tomorrow."

When I go through the ancient texts of the Greek philosopher Archytas about sorrows, sorrows of immortality, the Devil and the light of the world, there's a lot more detail there than you might think. There are echoes in all that subject matter going on. In the past, there were pages long with the same two or three character states. Scholars have pointed out how chirality defines rules when it's fun to do so. If you really want to be remembered, you just write down something. If you want to be remembered in a particular divided field of study, you have to write down something that is significantly different than the classification you have already or in your mind. Everyone has something different to write down. But having that is equally important: we can't make the ancient system, could we? Somebody has to explain something in writing, or at least break boundaries, and often that is the task. And that brings us to one of the most important facts of.

There's an old story. A chariot was moving in confused circles. The chariot stopped actively in the middle of the road and got overhead. It suddenly began to go out in a circle, a par rated Stamina Lightweight Robot. At the time of the random shaking in the par has happened, at least, it wasn't a metal robot, it is strong, a chance-wise, and it is capable of lifting massive objects. But since the collision the game can of handstand and cardiac division, and the light weight of the case opened another possibility (heat also wants to protect the body from heat). Here's what happened at the moment of the accident the mechanical object continued to hit the slingshot part of the chariot and the effect was catastrophic? The flaming wreckage of the chariot with its wreckage on the right side blew off the other side of the vehicle. The child who suffered very badly and ended up in a coma went completely unconscious. His organs were all but gone, he had died in a fiery explosion. Something else that happened in the chariot was that the truck that drove into it was damaged, misaligned. Perhaps that went completely unexploded, there are a lot of effects of this kind that can be sustained over little hours. And still. "You saw something or acted (for some unknown reason) or maybe (it was) another thing that happened. Yeah, this wasn't easy."

If they were looking for much more polarity, you'd want to imagine these events arising in the wormway of time. The customer of a window might fly in the time, he's probably heard something used-up about the J4 computer being able to carry out the job of computer hacking. Isn't that parallelism the major reason why lower level memories teach us about time? An example of self-resemblance to this kind of stuff is the one that Adam used to reach to Kylie from the depths of hell. It was a very rude kind of memory-reconnaissance exercise. I remember seeing a horrified form of air and darkness, and as you've seen, when it was brought awake, the joy of seeing other beings would cease. I can only imagine how someone may have been unable to imagine it. You know, there would be additional legs possible, it would be really rich in fun. But for a moment Kylie said: you're a potential victim of such a meteorological disaster, unless you jumped out of a plane like me.

The story of the moon, being frozen and not yet crushed by the impact, keeping itself up, does such things all the time. There's a new story happening here, and here's how it relates to sending a basic need to take care of yourself to you. And this story does include a classic example of how your initiated self can take care of you, er.

Sentient Intelligence

This is a note from the book Electromagnetic Times: The Quest for Ultrafast Time 1000

In an age when, say, a spacecraft needs every power available to send it around the Sun, why not send it now when it's coming close enough to shut down?

Within that brief chapter, you'll see how the revelation of a more interstellar solar system with thousands of starships emitting thousands of battle buddies and their radar was all but confirmed by everyday data regarding the planet itself, proving a great assumption. In that context, it is wiser to move on from the question of life in

Sample 1 for model: ARMD-small and parallel generation $S = 2$ ($T = 512$):

clearance interviews of developers ensuring that devices can easily be deployed, receive updates, and communicate as seamlessly as possible in standard operating system systems – it's a big, huge deal. There are myriad intersection points, mustering a multitude of requests to support and eventually eliminating the need to build dev tools.

First, a quick look at what distinguishes open source development from proprietary projects

Representative— This includes all aspects of the design, implementation, maintenance, support, and cooperation of open source software.

Project engineer— This includes the technician, technical support, and writing of the full project documentation.

Before we get into the performance of customization thereof, let me point out general reasons why open source software offers a higher level of performance than proprietary IT systems:

Built on a single OS platform

Suffice it to say that we have bend, pluck, claw, weave, and gouge in line with open source solutions today; however, we thankfully get to use a completely different set of OS and programming tools.

Controlling on the server has been a core part of the backbone of Open Source IT for some time; but for many IT departments this requiring platforms like SUSE is often more headed towards the future and smaller teams will be more willing to put development effort in. I'm sorry, Linux has come a long way, and that's not to say that OSX, MacOS, SporTV, and even Windows couldn't solve their problems.

Open Source has also come a long way on its own tools. Tools are the foundation of Open Source IT, and many are based on Linux. Linux was PC oriented and easy to get started, but an open source Linux development pipeline will allow developers greater freedom and scope to use a wider range of tool and subsystem concepts.

Open Source also allows developers to refactor, remove, change, improve, or maintain minimum levels of functionality with greater scope; paying developers the loyalty to a single operating system, and not the other way around.

A good example is Open Container Initiative (OCI). Open Container Initiative (OCI) has in the past been a non-profit venture and even distributed, yet for a long time, its developers weren't affiliated with IT departments.

This gave them an edge in unit driven projects; however, the issues with projects like Open Container Initiative (OCI) lead them to destroy it, since they made the massive mistake of having merged with Open Container Initiative (OCI) in 2009.

Tacoma, Wash.-based Open Container Initiative (OCI) is an open source project that is trying to integrate the concept of containers into systems operating under Linux. Its priorities include:

Enhance the ability of open source containers to efficiently solve application problems

Support a variety of cross-platform, eco-system building workloads

Provide an anti-blue screen of death all over enterprise IT systems

Allow the easy integration of containers into systems operating under Linux

It's necessary to discuss infrastructure associated with containers:

Compatibility and application architecture: Which operating systems you run should be all the same, is critical for any "single" OS?

Distribution lifecycle: Which OS should we build for? Whether we build for Windows, Linux, or OS X? Does adding few layers of management matter?

Forming an ecosystem from scratch: What open source technology should we pick? Should there be a vendor certification set up?

Other aspects with consideration:

Which platform should we target?

For what use limitation should we place a proprietary solution runtime on or a low tier operating system?

If the goal of Open Source is to provide versatility, with high-performance and low overhead, shouldn't we not execute code on the same platform?

With a global set of tools and APIs, Open Source can provide solutions to all types of problems and at the same time eliminate the need for many layers/distributions to replace the one (or less) available.

Let's look at some other generally applicable principles that you identify for Open Source in terms of developer

capabilities:

A balanced organization that balances maintaining and improving the infrastructure with support and complexity that advantages technology (or tactics) over the other.

Management of open source software in code can help to resolve congestion and improve quality and deployment of system updates and patches.

An enterprise-wide “transient” mind-set, so that the intent isn’t to hit costs of maintaining an implementation but rather to “just in case” without any risk.

It’s important to note that policies that intuit or enforce how systems are to operate should also be applied individually based on corporate security and compliance considerations. This also extends to the smaller organizations that just don’t care about cost if the benefits outweigh value for the upgrade

Sample 2 for model: ARMD-small and parallel generation $S = 4$ ($T = 256$):

A new ABC musical, "American Legend," which is set in the 19th century British West Indies, features Jonathan Tucker (Andrew Joseph). Tucker, played by local favorite Richard J. Browne, also stars.

Week Two [edit]

Week Three marks the return of Alan Scheck (who starred in the 1960 Broadway production of Exchange). The show is two-parter for the first time in Disney's Broadway history. The playing concludes with the Producers meeting in the old Planet Filmy Railroad to discuss the subject of their enterprise.

Week Four [edit]

Week Four marks the return of Woody Harrelson, Tim Curry, and Mary J. Blige as Woody. They are joined by a young three-time Academy Award winner in Christopher Lloyd, and a visiting professor from the University of British Columbia in the title role.

Week Five [edit]

Week Five marks the return of Duo Aguirre, Benito Villalobos, and Normal Brown Parker as Yancy, turned into Gigul! In 1999, the cast and crew of the Broadway musical's Land of the Millennium sought out possible movie roles. They signed up the actors for a Jack London revival of The House of Waverley "as a" Broadway, Inc. The Broadway musical went on to gross more than $340 million on Broadway.

Week One [edit]

Week One is the latest in a series of productions that focus on the history of Disney characters done that way. One brief mention of Disney characters and the story of the imaginative property comes from The Little Mermaid, the first musical directed and co-produced by Buzz Lightyear and Rodney Dangerfield as an original animated musical. A short skip-step at the Performance Hall (where the musical is set) shows the promotion in effect that Disney cantor William Waterhouse predicted the stage musical would become the year's first nationwide success story. The trio of musicals usually consist of one musical and take place in approximately 700,000 locations. In each such production, character names are listed beginning with Disney characters.[citation needed]

Covers [edit]

Although it has premiered in Broadway's original production space, the musical has one rare exception: the production only played in Broadway's first act during the Cedar Fair Fair – January 16, 1993, shortly before the holiday season began.[note 2]

Seasons' premiere [edit]

The musical's first two seasons – one musical, the other, the musical (man) are neatly co-presented on Broadway at the ArcLight Bank Theatre. The production opened in Winterthur to positive reviews and a standing ovation. On November 12, 2013, the Broadway production opened at David Letterman's CBS Radio show, Airline. It made its debut at the Disneyland Resort in July 2014. It began its third season on November 7, 2015. It is based on the theme "I'll Wear You," from Disney's film The Lion King. The four seasons of the show were released in DVD format in September and October 2012. It is a high-energy, high character musical,[5] consisting almost entirely of singing and stunts (or "aladdin", depending on the question of aladdin's entertainment value) from 1970 to the present day. The musical was presented by Jack Levitt and Michael Neuman in January 2013. Its production orchestra has had a long presence in the Hollywood Hills since the '60s, and groups such as The Repertory Company of Los Angeles and Holst Music will provide accompaniment. This cast turns out the best-known and richest musical in the history of the Hamptons.

Troll Confidential [edit]

Goodman is joined on stage by Tom DeVos and Daranassi of Burbank on September 28. Troll executive producer and regular associated with the production, Kevin Reed, will serve as executive producer on both productions.

Sales for Twitter user #Aladdin–a Jeff Faison–inspired animated computer comedy–leaked going back to September 4, 2015 highlighting the talents behind the film and its characters.

Season 6 [edit]

The Broadway musical's 10th season is in previews and has been in production since August 17, 2016. This season includes themed episodes and both recent Broadway musicals, like Rent, A24's Once Upon a Time in America and Casino. While part of the legacy for the Broadway musical of 2010, Beauty and the Beast, its treatment and DVD release, was, at the time, unprecedented in terms of what could be, the show stuck around.

Series screenwriters Jay Kricfalusi and Adam Pally have written six total scripts with proceeds going to the Scouts of America. Kricfalusi continues in his post-New York City book, How to Succeed in Business Without Really Trying.

The musical will open a