BLOCKGEN: FLEXIBLE BLOCKWISE SEQUENCE MODELING WITH HYBRID SAMPLERS

Justin Deschenaux[1*] **Caglar Gulcehre**[1,2]

[1]EPFL, Lausanne, Switzerland [2]Microsoft AI

ABSTRACT

Block Diffusion Models (BDMs) accelerate discrete diffusion by generating token blocks in parallel while supporting KV caching. However, existing BDMs are typically trained with a single, *fixed* block size, limiting the trade-offs at inference time. Moreover, most BDMs use masked diffusion, where tokens cannot be revised once generated, limiting quality in parallel decoding scenarios. We introduce *BlockGen*, a general framework for blockwise sequence modeling that trains a single model over a *set* of block sizes and is compatible with arbitrary block conditionals. In this work, we instantiate BlockGen with *uniform-state* discrete diffusion within each block. BlockGen achieves improved likelihood compared to fixed block-size training and higher sample quality with fewer denoising steps. Training on multiple block sizes enables hybrid samplers that combine autoregressive and diffusion predictions, substantially improving over pure block-by-block generation while preserving KV caching.

1 INTRODUCTION

Autoregressive (AR) Transformers are the dominant paradigm for sequence modeling, which powers modern language models through next-token prediction (Bengio et al., 2000; Touvron et al., 2023; Meta, 2024; Google, 2025; OpenAI, 2024). Although the Transformer architecture has enabled large-scale training (Vaswani et al., 2023), standard causal decoding under the autoregressive factorization still generates tokens one by one, limiting throughput and increasing latency.

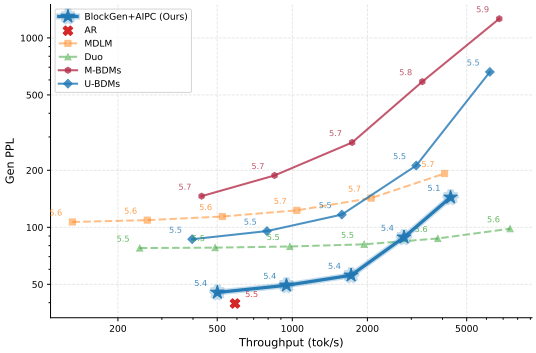


Figure 1: **Generative Perplexity and Unigram Entropy vs Throughput**. BlockGen pushes the Pareto frontier of Masked and Uniform block diffusion (BDMs) and approaches the AR performance on OpenWebText. All models use a block size 16 and BlockGen generates the first 256 tokens autoregressively.

In contrast, diffusion language models (DLMs) and related discrete flow-matching methods iteratively refine a noisy sequence using a bidirectional denoiser, potentially updating multiple tokens in parallel (Sohl-Dickstein et al., 2015; Austin et al., 2023; Campbell et al., 2022; 2024; Gat et al., 2024; Sahoo et al., 2024; Shi et al., 2025; Ou et al., 2025; Lou et al., 2024). Furthermore, Nagarajan et al. (2025) have shown that multi-token approaches, such as diffusion language models, produce a more diverse and original output compared to autoregressive models. Hence, DLMs enables a flexible quality-compute trade-off by varying the number of denoising steps.

Unfortunately, full-sequence discrete diffusion is expensive in inference. Because the denoiser is bidirectional, activations must be recomputed at every denoising step, and exact KV caching (Pope et al., 2022) is not possible (Deschenaux & Gulcehre, 2024). Block Diffusion Models (BDMs)

*Correspondence to justin.deschenaux@epfl.ch

(Arriola et al., 2025a; Wu et al., 2025; Arriola et al., 2025b) enable caching by generating left-to-right in blocks. Each block is sampled with discrete diffusion conditioned on a clean prefix, allowing for parallel updates within the block while reusing the keys/values of prior blocks across denoising steps.

Despite these gains, existing BDMs come with limitations: **(1) Fixed parallelism.** Training with a single block size restricts the space of possible sampling algorithms to a fixed level of parallelism. **(2) Poor few-step quality.** Most BDMs use masked diffusion, which cannot re-visit previously unmasked tokens and performs poorly at low NFEs (Sahoo et al., 2025a; Schiff et al., 2025). **(3) Inconsistent generation.** Because the denoiser predicts token-wise marginals for close, correlated tokens, BDMs tend to generate values that are unlikely under a joint AR factorization. This is especially problematic when generating early blocks with limited context.

Contributions Motivated by these observations, we introduce *BlockGen*, a general framework for blockwise sequence modeling over *variable* block sizes. (1) We show that training over multiple block sizes improves likelihood over fixed block size BDMs. (2) We instantiate BlockGen with uniform-state diffusion and show that it improves the quality of the sample while allowing self-correction. (3) We demonstrate the flexibility of BlockGen through sampling algorithms that vary the block size at the inference time.

Why variable block sizes? Training with variable block size addresses a fundamental weakness of traditional block diffusion models. Generating large blocks maximizes parallelism but require predicting more tokens jointly. Small blocks improve consistency but sacrifice throughput. Training a single model over multiple block sizes enables flexible inference strategies, such as using small blocks when context is limited and increasing parallelism as the sequence grows. This also enables hybrid samplers that combine AR and diffusion predictions.

Organization. We review background material in Sec. 2, introduce BlockGen in Sec. 3, present hybrid sampling strategies in Sec. 4, and evaluate on density estimation Sec. 5 and generation quality in Sec. 6.

2 Background

Notation We represent the vocabulary as one-hot vectors $\mathcal{V} := \{\mathbf{v} \in \{0,1\}^{|\mathcal{V}|} : \|\mathbf{v}\|_1 = 1\}$. A sequence $\mathbf{x} \in \mathcal{V}^L$ consists of L tokens, with $\mathbf{x}^\ell$ denoting its ℓ-th element. We write $\Delta^{|\mathcal{V}|}$ for the probability simplex and $\mathrm{Cat}(\cdot; \mathbf{v})$ for the categorical distribution with parameter $\mathbf{v} \in \Delta^{|\mathcal{V}|}$. We denote by $\boldsymbol{\pi} \in \Delta^{|\mathcal{V}|}$ a fixed prior, $\mathbf{1}$ the all-ones vector, L the sequence length, and L' the block size.

Autoregressive Language Modeling Autoregressive (AR) language models factorize the distribution over sequences $\mathbf{x} \in \mathcal{V}^L$ as $p_\theta(\mathbf{x}) = \prod_{\ell=1}^{L} p_\theta(\mathbf{x}^\ell \mid \mathbf{x}^{<\ell})$, where $\mathbf{x}^{<\ell}$ denotes the prefix before position ℓ. The AR decomposition enables efficient training but requires sequential generation with one forward pass per token.

2.1 Discrete Diffusion Models

Discrete diffusion models (Sohl-Dickstein et al., 2015; Austin et al., 2023; Campbell et al., 2022; Lou et al., 2024) define a sequence of increasingly noisy distributions $(q_t)_{t\in[0,1]}$ that interpolate from the data distribution q_{data} to a factorized noise distribution $\prod_{\ell=1}^{L} \mathrm{Cat}(\cdot; \boldsymbol{\pi})$. The noisy latents $\mathbf{z}_t \sim \prod_{\ell=1}^{L} q_t(\cdot|\mathbf{x}^\ell)$ are obtained through Markovian transitions applied independently across positions in the sequence. In this work, we focus on *interpolating discrete diffusion processes*, whose marginals $q_t(\cdot|\mathbf{x}^\ell)$ take the form:

$$\mathbf{z}_t^\ell \sim q_t(\cdot|\mathbf{x}^\ell; \alpha_t) = \mathrm{Cat}\big(\cdot\,; \alpha_t \mathbf{x}^\ell + (1-\alpha_t)\boldsymbol{\pi}\big), \tag{1}$$

where $\alpha_t \in [0,1]$ is a monotonically decreasing *noise schedule* with $\alpha_0 \approx 1$ and $\alpha_1 \approx 0$. (1) defines the *forward process*, which progressively corrupts $\mathbf{x}$ into a sample from the prior $\boldsymbol{\pi}$.

Generative Process To generate samples, diffusion models define a *generative process* p_θ that reverses the corruption in (1). Given a time-discretization $0 = t_0 < t_1 < \cdots < t_T = 1$, the generative process factorizes over the reverse trajectory $p_\theta(\mathbf{z}_{t_0}, \ldots, \mathbf{z}_{t_T}) = p(\mathbf{z}_{t_T}) \prod_{i=1}^{T} p_\theta(\mathbf{z}_{t_{i-1}} \mid \mathbf{z}_{t_i})$, where $p(\mathbf{z}_{t_T}) = \prod_\ell \mathrm{Cat}(\cdot; \boldsymbol{\pi})$ is the sequence-level prior. Each reverse transition is constructed by substituting the unknown $\mathbf{x}$ in the true posterior $q_{s|t}(\cdot \mid \mathbf{z}_t, \mathbf{x})$ with the output of a learned denoiser $\hat{\mathbf{x}}_\theta$, as $p_\theta(\mathbf{z}_s \mid \mathbf{z}_t) = \prod_{\ell=1}^{L} q_{s|t}\big(\cdot \mid \mathbf{z}_t^\ell, \hat{\mathbf{x}}_\theta^\ell(\mathbf{z}_t, t)\big)$. The denoiser $\hat{\mathbf{x}}_\theta : \mathcal{V}^L \times [0,1] \to (\Delta^{|\mathcal{V}|})^L$ is trained by minimizing the standard diffusion Negative Evidence Lower Bound (NELBO). The form of the posterior $q_{s|t}$ depends on the choice of prior $\boldsymbol{\pi}$. We describe two instantiations below.

Masked Diffusion Models (MDMs) MDMs (Sahoo et al., 2024; Shi et al., 2025; Ou et al., 2025) use a *masked prior*, where $\boldsymbol{\pi} = \mathbf{m} \in \mathcal{V}$ is the one-hot representation of a special token [MASK] . In the forward process (1), each token is preserved or replaced by the token [MASK] . Once masked, tokens remain in this absorbing state for the rest of the trajectory. This behavior carries over to the reverse process. The posterior distribution $q_{s|t}^{\mathrm{MDM}}$ for $0 \le s < t \le 1$ can be derived via Bayes' rule:

$$q_{s|t}^{\mathrm{MDM}}(\cdot|\mathbf{z}_t^\ell, \mathbf{x}^\ell) = \begin{cases} \mathrm{Cat}\left(\cdot; \frac{\alpha_s - \alpha_t}{1-\alpha_t}\mathbf{x}^\ell + \frac{1-\alpha_s}{1-\alpha_t}\mathbf{z}_t^\ell\right) & \text{if } \mathbf{z}_t^\ell = \mathbf{m}, \\ \mathrm{Cat}(\cdot; \mathbf{x}^\ell) & \text{otherwise.} \end{cases} \tag{2}$$

The learned generative process replaces $\mathbf{x}$ with the predictions of a denoiser $\hat{\mathbf{x}}_\theta$: $p_\theta(\mathbf{z}_s \mid \mathbf{z}_t) = \prod_\ell q_{s|t}^{\mathrm{MDM}}\big(\cdot \mid \mathbf{z}_t^\ell, \hat{\mathbf{x}}_\theta^\ell(\mathbf{z}_t, t)\big)$, where $\hat{\mathbf{x}}_\theta : \mathcal{V}^L \times [0,1] \to (\Delta^{|\mathcal{V}|})^L$. A key limitation, evident from (2), is irreversibility: once a token is unmasked it cannot be changed, hence mistakes cannot be corrected and compound over time. See (15) for the expression of the NELBO.

Uniform-State Diffusion Models (USDMs) USDMs (Schiff et al., 2025; Sahoo et al., 2025a) use a *uniform prior* $\boldsymbol{\pi} = \mathbf{1}/|\mathcal{V}|$. Unlike MDMs, this choice allows tokens to transition between any states throughout the generative process, enabling *self-correction* of earlier mistakes. The posterior distribution $q_{s|t}^{\mathrm{USDM}}$ takes the form:

$$q_{s|t}^{\mathrm{USDM}}(\cdot|\mathbf{z}_t^\ell, \mathbf{x}^\ell) = \mathrm{Cat}\Big(\cdot; \frac{|\mathcal{V}|\alpha_t(\mathbf{z}_t^\ell \odot \mathbf{x}^\ell) + (\alpha_{t|s} - \alpha_t)\mathbf{z}_t^\ell + (\alpha_s - \alpha_t)\mathbf{x}^\ell + (1-\alpha_{t|s})(1-\alpha_s)\mathbf{1}/|\mathcal{V}|}{|\mathcal{V}|\alpha_t\langle \mathbf{z}_t^\ell, \mathbf{x}^\ell\rangle + 1 - \alpha_t}\Big) \tag{3}$$

where $\alpha_{t|s} = \alpha_t/\alpha_s$. Due to their self-correction capability, USDMs significantly outperform MDMs in few-step generation (Sahoo et al., 2025a) and are better suited for guided generation (Nisonoff et al., 2024; Schiff et al., 2025). See (16) for the expression of the NELBO.

Predictor-Corrector Samplers Predictor-Corrector (PC) samplers have previously been explored as an alternative to ancestral sampling (Song et al., 2021; Campbell et al., 2022; Gat et al., 2024; Campbell et al., 2024; Wang et al., 2025a; Anonymous, 2026). PC samplers alternate (or combine) a *predictor* updates that moves from diffusion time $t \to s$, where $s < t$, with *corrector* steps that injects noise. With MDMs, corrector steps enable re-masking and thus correcting earlier errors. For USDMs, re-injecting random tokens can also improve quality (Anonymous, 2026). A simple PC scheme first samples a clean proposal $\tilde{\mathbf{x}}$ from the denoiser then re-noises to time $s < t$

$$\tilde{\mathbf{x}}^\ell \sim q_{0|t}(\cdot \mid \mathbf{z}_t^\ell, \hat{\mathbf{x}}_\theta^\ell(\mathbf{z}_t, t)), \qquad \xrightarrow{\text{then}} \qquad \mathbf{z}_s^\ell \sim q_s(\cdot \mid \tilde{\mathbf{x}}^\ell; \alpha_s) = \mathrm{Cat}\big(\cdot; \alpha_s\tilde{\mathbf{x}}^\ell + (1-\alpha_s)\boldsymbol{\pi}\big). \tag{4}$$

2.2 Block Diffusion Models

Block diffusion models (BDMs; Han et al. (2023); Arriola et al. (2025a); Wu et al. (2025)) combine an autoregressive outer factorization with discrete diffusion conditionals. Let $\mathbf{x} \in \mathcal{V}^L$ and fix a block length $L' \le L$ such that $B := L/L'$ denotes the number of blocks. Let $\mathbf{x}$ be partitioned into B contiguous blocks $\mathbf{x}^1, \ldots, \mathbf{x}^B$, such that $\mathbf{x}^b = [\mathbf{x}^{(b-1)L'+1}, \ldots, \mathbf{x}^{bL'}]$. With BDMs, the likelihood factorizes as

$$p_\theta(\mathbf{x}) = \prod_{b=1}^{B} p_\theta(\mathbf{x}^b \mid \mathbf{x}^{<b}), \tag{5}$$

where each conditional $p_\theta(\mathbf{x}^b \mid \mathbf{x}^{<b})$ is parameterized by a discrete diffusion model (Sec. 2.1), with $\mathbf{x}^{<b}$ denoting the first $b-1$ blocks. Since (5) is autoregressive over blocks, BDMs are often described

as semi-autoregressive (semi-AR). For block b, the diffusion forward process (1) is applied only to tokens in $\mathbf{x}^b$, yielding noisy latents $\mathbf{z}_t^b$. The corresponding reverse transitions, conditioned on $\mathbf{z}_t^b$ and $\mathbf{x}^{<b}$, are given by $p_\theta(\mathbf{z}_s^b \mid \mathbf{z}_t^b, \mathbf{x}^{<b}) = \prod_{\ell=(b-1)L'+1}^{bL'} q_{s|t}(\cdot \mid \mathbf{z}_t^\ell, \hat{\mathbf{x}}_\theta^\ell(\mathbf{z}_t^b, \mathbf{x}^{<b}, t))$ Prior work (Arriola et al., 2025a; Wu et al., 2025) typically instantiates $q_{s|t}$ using the masked-diffusion posterior (2). Training minimizes the sum of NELBOs per-block.

KV Caching Arriola et al. (2025a) parameterize the denoiser $\hat{\mathbf{x}}_\theta$ with a Transformer using a *block-causal* attention mask (see Suppl. A.3 for details). The tokens in the block b attend bidirectionally within the block and causally to all the tokens in the preceding blocks $\mathbf{x}^{<b}$. At inference time, this enables block-wise KV caching (Pope et al., 2022). When generating block b, the keys/values for the already-generated prefix $\mathbf{x}^{<b}$ can be reused across all diffusion steps within the block.

3 BlockGen: Training a Mixture over Block Sizes

We define BlockGen in Sec. 3.1 and derive two likelihood bounds. Sec. 3.2 presents an efficient training procedure and a variance reduction scheme that improves the validation likelihood.

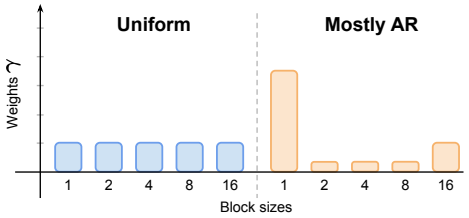


Figure 2: Block-size mixture weights $\boldsymbol{\gamma}$ we train with.

3.1 Mixture Formulation over Block Sizes

BlockGen is defined as a mixture over M block-size-specific densities. Let $\mathcal{S} = \{s_1, \ldots, s_M\} \subset \mathbb{N}$ be a set of block sizes (e.g., $\{1, 2, 4, \ldots, 2^{M-1}\}$) and $\boldsymbol{\gamma} \in \Delta^M$ a mixture weight. The BlockGen density is defined as

$$p_\theta^{\text{BlockGen}}(\mathbf{x}) = \sum_{i=1}^{M} \gamma_i \, p_\theta^{(s_i)}(\mathbf{x}), \tag{6}$$

where each component $p_\theta^{(s_i)}$ is a valid density factorized into blocks of size s_i. BlockGen is agnostic to the paradigm used to model the block conditionals. In this work, we instantiate BlockGen with uniform-state diffusion. In the case $L' = 1$, it reduces to AR modeling.

Likelihood bounds Assume each component $p_\theta^{(s_i)}$ admits either a tractable likelihood or *evidence lower-bound* (ELBO) $\mathcal{E}^{(s_i)}(\theta, \mathbf{x}) \le \log p_\theta^{(s_i)}(\mathbf{x})$. When the likelihood is tractable, we set $\mathcal{E}^{(s_i)} = \log p_\theta^{(s_i)}$. The mixture density (6) is then lower-bounded by

$$\log p_\theta^{\text{BlockGen}}(\mathbf{x}) \;\ge\; \underbrace{\log \sum_{i=1}^{M} e^{\mathcal{E}^{(s_i)}(\theta,\mathbf{x}) + \log \gamma_i}}_{\text{log-sum-exp bound (eval)}} \;\ge\; \underbrace{\sum_{i=1}^{M} \gamma_i \, \mathcal{E}^{(s_i)}(\theta, \mathbf{x})}_{\text{Mixture likelihood bound (train)}} . \tag{7}$$

Training (Sec. 3.2)**:** Optimize the mixture likelihood bound (with stratified block selection).
Evaluation (Suppl. C)**:** Report the log-sum-exp bound by computing all M component ELBOs.

Geometric Mean Parameterization An alternative *Geometric Mean Parameterization* (GMP) admits the mixture likelihood bound as ELBO via Hölder's inequality (Sec. D.2), though we do not use it in this work:

$$p_\theta^{\text{GMP}}(\mathbf{x}) \propto \prod_{i=1}^{M} \left(p_\theta^{(s_i)}(\mathbf{x}) \right)^{\gamma_i} . \tag{8}$$

3.2 Efficient Training with Stratified Block Sizes

We optimize θ by maximizing the ELBO, or equivalently, minimizing the bound on the negative log-likelihood (NLL). The gradient of the log-sum-exp bound takes the form:

$$\nabla_\theta \left[-\log \sum_{i=1}^{M} \gamma_i \, e^{\mathcal{E}^{(s_i)}} \right] = -\sum_{i=1}^{M} \omega_i \, \nabla_\theta \mathcal{E}^{(s_i)}(\theta, \mathbf{x}), \quad \text{where} \quad \omega_i = \frac{\gamma_i \, e^{\mathcal{E}^{(s_i)}(\theta,\mathbf{x})}}{\sum_{j=1}^{M} \gamma_j \, e^{\mathcal{E}^{(s_j)}(\theta,\mathbf{x})}}. \tag{9}$$

Computing the gradient in (9) is impractical: it requires evaluating M ELBOs, making training $M\times$ more expensive. Fortunately, the mixture likelihood bound admits a cheap unbiased estimator. We optimize the mixture likelihood bound during training and report the log-sum-exp bound at evaluation (Suppl. C). The mixture likelihood bound is an expectation over block sizes, $-\sum_{i=1}^{M} \gamma_i \mathcal{E}^{(s_i)}(\theta, \mathbf{x}) = \mathbb{E}_{i\sim \mathrm{Cat}(\boldsymbol{\gamma})}[-\mathcal{E}^{(s_i)}(\theta, \mathbf{x})]$, so we sample $i \sim \mathrm{Cat}(\boldsymbol{\gamma})$ at each step and compute $-\nabla_\theta \mathcal{E}^{(s_i)}(\theta, \mathbf{x})$.

Stratified block size selection When training on $D > 1$ GPUs, assigning different block sizes to each GPU via stratified sampling reduces variance and improves the final model while preserving unbiasedness (Kroese et al., 2011). Concretely, we partition $(0, 1]$ into M bins via the cumulative sums of $\boldsymbol{\gamma}$, sample $u \sim \mathrm{Uniform}(0, 1)$, and shift it by D evenly-spaced offsets (wrapping modulo 1) to obtain one block size per GPU. See Algo. 2 for pseudocode.

Denoising backbone Rather than learning separate models for each $p_\theta^{(s_i)}$, we use a single shared denoiser $\hat{\mathbf{x}}_\theta(\cdot, \cdot, t; L')$ that takes the block size L' as an additional input and sets the attention patterns accordingly. We follow the Diffusion Transformer (DiT) architecture of Arriola et al. (2025a), with one modification (see Suppl. A.3): we optionally replace their block-causal attention with standard causal attention over the prefix, which enables sharing the KV cache across all block sizes during sampling.

4 Hybrid Sampling Strategies

A denoiser trained on multiple block sizes enables flexible block selection at inference. In particular, the AR mode (L'=1) can verify token consistency. We present two strategies that combine AR and diffusion predictions.

AR Seeding Early blocks have little context, so parallel decoding is prone to inconsistencies that propagate to later blocks. A simple strategy consists of generating the first k tokens autoregressively before switching to block-by-block diffusion.

4.1 AR-Informed Predictor-Correctors (AIPC)

Within a block, the diffusion denoiser predicts independent per-position marginals, which can yield inconsistent tokens. AIPC addresses this by alternating standard diffusion steps with *informed steps* that re-inject noise at low-confidence positions identified by the AR mode.

Informed steps At diffusion time t, let $p^{\mathrm{D}} = \hat{\mathbf{x}}_\theta(\mathbf{z}_t^b, \mathbf{x}^{<b}, t; L')$ be the denoiser output. A standard step samples $\mathbf{z}_s^b \sim q_{s|t}(\cdot \mid \mathbf{z}_t^b, p^{\mathrm{D}})$. An informed step instead (i) samples a clean proposal $\tilde{\mathbf{x}}^b \sim q_{0|t}(\cdot \mid \mathbf{z}_t^b, p^{\mathrm{D}})$, (ii) computes AR predictions $p^{\mathrm{AR}} = \hat{\mathbf{x}}_\theta(\tilde{\mathbf{x}}^b, \mathbf{x}^{<b}; L'{=}1)$, (iii) uses a scoring function g to evaluate the quality of each token:

Table 1: Validation perplexity (Val. PPL) on OWT after 1M training steps. **Takeaway:** BlockGen closes the gap between block diffusion and autoregressive models, achieving 17.5 PPL vs. 16.7 for AR. †Adapted from MDLM checkpoint (850k steps). ‡From Sahoo et al. (2025a). **Bold**/underline: best/second-best block diffusion. Training Masked BDMs with unweighted CE improves likelihood.

Model	Val. PPL
Autoregressive	
Transformer	16.7
Sequence Diffusion (Masked)	
SEDD Absorb‡ (Lou et al., 2024)	24.1
MDLM‡ (Sahoo et al., 2024)	23.2
Sequence Diffusion (Uniform)	
SEDD Uniform‡ (Lou et al., 2024)	29.7
UDLM‡ (Schiff et al., 2025)	27.4
Duo‡ (Sahoo et al., 2025a)	25.2
BDM (single-block, $L' = 16$)	
BDM† (Arriola et al., 2025a)	22.3
Masked (CE)	21.6
Uniform (ELBO)	23.6
BlockGen (ours, $L_{\max} = 16$, uniform $\boldsymbol{\gamma}$)	
Masked	**17.5**
Uniform	<u>18.5</u>

$g(p^{\mathrm{D}}, p^{\mathrm{AR}})$, and (iv) re-noises the k lowest-scoring positions, where $k = \texttt{round}((1-\alpha_s)L')$, to match the noise level at time s. We find that at matched NFEs, AIPC outperforms samplers that do not use the AR predictions.

Step schedule Observe that informed steps use two forward passes (one with $L' = 1$, one with $L' > 1$). We use informed after k_{init} pure diffusion steps, and every $r \geq 1$ update afterwards (Sec. 6.2). In Algo. 1, GUIDED(i) is true when step i is an informed step.

5 DENSITY ESTIMATION

First, we train single-block baselines for both masked and uniform BDMs (Sec. 5.1). Second, we show that joint training over multiple block sizes significantly improves likelihood, thanks to the AR component (Sec. 5.2).

Setup We evaluate on LM1B (Chelba et al., 2014) and OpenWebText (OWT; Gokaslan & Cohen (2019)). All models are 170M-parameter Diffusion Transformers (Peebles & Xie, 2023) with RoPE (Su et al., 2023), trained with the mixture likelihood bound ((7)) and a linear noise schedule. We first ablate at 250k steps and train the best models for 1M steps. The models trained for 1M steps use causal attention over clean tokens (Sec. 3.2, Suppl. A.3). We report the log-sum-exp bound for validation. See Suppl. A for more details.

5.1 SINGLE-BLOCK BASELINES

We compare different training objectives for masked and uniform single-block-size BDMs after 250k training steps.

Masked BDMs Table 5 shows the validation perplexity for single-block BDMs on LM1B and OWT after 250k training steps. For Masked BDMs, Arriola et al. (2025a) use a costly variance reduction scheme that optimizes the noise schedule every 5k steps over the validation set. We find that training with unweighted cross-entropy matches their performance on OWT and slightly improves it on LM1B, without the overhead. We therefore use the unweighted CE for all masked models, which recent work showed optimizes the true NELBO (Shi & Titsias, 2025).

Uniform BDMs For uniform-state BDMs, we experimented with the unweighted CE on noisy positions as an alternative to the NELBO. The CE performs well on LM1B but underperforms on OWT (except at block size ≤ 4), so we train USDMs with the NELBO for block sizes > 1 in all later experiments. Consistent with prior work on full-sequence diffusion (Schiff et al., 2025; Sahoo et al., 2025a), uniform-state BDMS slightly underperform their masked counterpart in likelihood. At the same time, Uniform BDMs outperform full-sequence USDMs such as Duo (Sahoo et al., 2025a).

Training with $L' = 1$ For block size 1, we always train with CE for both uniform and masked BDMs. Rather than applying the diffusion forward process, we always corrupt the input token and predict the clean target, following Arriola et al. (2025a).

Algorithm 1: AIPC Sampler (one block)

Require: Prefix $\mathbf{x}^{<b}$, block size L', denoiser $\hat{\mathbf{x}}_\theta$, scoring function g

$\mathbf{z}^b \sim \text{Prior}$
for $i = 0$ **to** $T-1$ **do**
  $t \leftarrow (T-i)/T, \quad s \leftarrow (T-i-1)/T$
  $p^{\mathrm{D}} \leftarrow \hat{\mathbf{x}}_\theta(\mathbf{z}^b, \mathbf{x}^{<b}, t; L')$
  if GUIDED(i) **then**
    $\tilde{\mathbf{x}}^b \sim q_{0|t}(\cdot \mid \mathbf{z}^b, p^{\mathrm{D}})$
    $p^{\mathrm{AR}} \leftarrow \hat{\mathbf{x}}_\theta(\tilde{\mathbf{x}}^b, \mathbf{x}^{<b}; L'=1)$
    $k \leftarrow \text{round}((1-\alpha_s) \cdot L')$
    $\mathbf{m} \leftarrow \text{BOTTOMK}(g(p^{\mathrm{D}}, p^{\mathrm{AR}}), k)$
    $\mathbf{z}^b \leftarrow \text{RENOISE}(\tilde{\mathbf{x}}^b, \mathbf{m})$
  else
    $\mathbf{z}^b \sim q_{s|t}(\cdot \mid \mathbf{z}^b, p^{\mathrm{D}})$
  end if
end for
return $\mathbf{z}^b$

5.2 JOINT TRAINING IMPROVES LIKELIHOOD

Training BlockGen as a mixture over block sizes (Sec. 3.2) significantly improves the validation likelihood compared to single block-size models. Table 7 presents the validation perplexity of each

component, showing that the AR component explains the improvement. Notably, stratified block size selection (Sec. 3.2) outperforms using a single block size per gradient step. Prior work showed that USDMs yield better text quality than MDMs due to self-correction (Sahoo et al., 2025a; Anonymous, 2026). We therefore focus on Uniform BDMs, training only a few masked variants.

Selecting the weights $\boldsymbol{\gamma}$ BlockGen requires choosing the weight vector $\boldsymbol{\gamma}$ in (7). We consider two strategies. **Uniform** assigns equal mass to all block sizes ($\gamma_i = 1/n$). **Mostly AR** sets $\gamma_1 > \gamma_n > \gamma_2 = \cdots = \gamma_{n-1}$, prioritizing AR while training on all block sizes (Figure 2). Mostly AR improves the likelihood (Table 7), but may hurt generation quality (Table 6). Hence, we use the Uniform $\boldsymbol{\gamma}$ in Sec. 6. We first identify why standard block diffusion struggles with sample quality (Sec. 5.3), then show that AR seeding improves quality (Sec. 6.1), and finally demonstrate that AIPC further improves quality by leveraging AR predictions (Sec. 6.2).

Experimental setup We train models from scratch for 1M steps with Uniform and Mostly AR weight vectors $\boldsymbol{\gamma}$. We use powers-of-two block sizes $\{1, 2, \ldots, 2^{M-1}\}$ for $M \in \{5, 6, 7\}$. The baselines are AR, MDM, Duo, and single-block BDMs. Sample quality is measured with the generative perplexity (Gen. PPL) under GPT-2 Large (Radford et al., 2019), and diversity using the unigram entropy (Dieleman et al., 2022; Sahoo et al., 2024; 2025a). We cast logits to 64-bit precision before sampling (Zheng et al., 2025). In diffusion mode, we use the posteriors in (2) and (3), no temperature scaling or nucleus annealing.

5.3 Why Standard Block Diffusion Struggles

Standard block diffusion with ancestral sampling underperforms full-sequence diffusion (Table 8). Even when using L forward passes per block, masked BDMs exhibit higher generative perplexity than AR and MDM. The core issue is that BDMs generate highly correlated tokens in parallel. Since the denoiser predicts per-token marginals instead of joints, BDMs risk sampling inconsistent tokens. Full-sequence diffusion is less affected because it updates tokens that may be far apart and thus less correlated. Notably, Uniform BDMs suffer less from this than Masked BDMs.

6 Improving Sample Quality with Hybrid Sampling

Why Uniform BDMs Closes the Gap? Replacing full-sequence with block-wise diffusion affects masked and uniform models differently. For MDMs, forcing left-to-right generation only adds constraints compared to any-order. For USDMs, full-sequence denoising is harder because the model cannot distinguish clean from noisy tokens. Block-wise generation partially addresses the ambiguity, since the prefix is always clean.

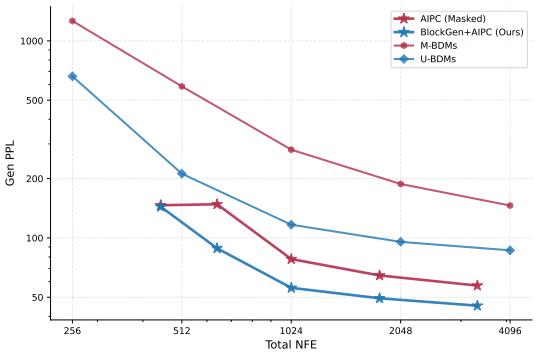


Figure 3: Generative Perplexity (Gen. PPL) of AIPC with AR seeding ($k = 256$), compared to regular BDMs. AIPC consistently improves sample quality by using AR predictions to identify and re-inject noise into low-quality tokens. As shown in Table 2, BlockGen has a slightly higher throughput.

6.1 AR Seeding Improves Quality

When sampling block by block, BlockGen produces lower-quality samples than single-block-size models. This is expected since a model trained on a single block size can dedicate all its capacity to that granularity. The same trade-off appears in validation perplexity, where evaluating BlockGen at block size L' underperforms models trained solely for L'. However, a shared model across block sizes enables alternative sampling strategies (Sec. 4).

AR Seeding We vary the prefix length $k \in \{64, 128, 256\}$ using BlockGen trained with uniform $\boldsymbol{\gamma}$ up to block size 16, 32, 64. Longer prefixes improve quality at slightly higher cost. With $k = 256$, AR

seeding allows masked BlockGen to outperform its single-block-size counterpart (Table 8, Table 9). Uniform BlockGen, however, still underperforms even with $k = 256$, motivating AIPC.

6.2 AIPC Beats Ancestral Sampling

As described in Sec. 4.1, AIPC interleaves AR predictions with diffusion during block generation. We combine AIPC with AR seeding, first sampling k tokens autoregressively, then using AIPC during block-by-block generation. For fair comparison, we match the total *number of function evaluation* (NFE) with other samplers. For example, with a budget of 8 passes per block, AIPC uses 8 steps total, counting both AR and diffusion calls.

Key Questions AIPC has several hyperparameters. We investigate three questions: (1) How should AIPC steps be scheduled (when to begin and how often)? (2) How do block size and prefix length affect quality? (3) Does AIPC outperform simpler alternatives such as random noise injection (Campbell et al., 2022; Gat et al., 2024; Campbell et al., 2024; Wang et al., 2025a; Anonymous, 2026) or noise injection based on predictions with $L' > 1$ (i.e. not using the AR component)? Complete results and settings are reported in Suppl. B.

Q1: Scheduling AIPC Steps We parameterize AIPC schedules by the first informed step k_{init} and the period pc (one informed step every pc denoising steps). Across settings, applying AIPC at low noise levels is most effective. With this choice, AIPC consistently improves over pure diffusion updates at matched NFE.

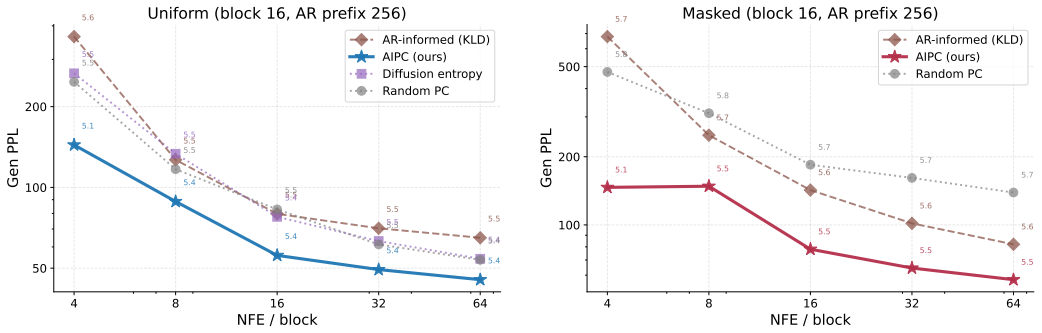
Q2: Block Size and Prefix Length Sampling with Larger block sizes L' substantially improve the throughput, and AIPC is able to maintain comparable Gen. PPL as the block size grows (Table 2). Recall the prefix length k controls the number of AR-seeded tokens. Without AIPC, reducing k causes a clear quality drop (Table 9). In contrast, AIPC is much less sensitive to shorter prefixes Table 2.

Q3: Comparison with Simpler Samplers At matched NFE per block, AIPC outperforms simpler predictor-corrector variants. Figure 4 compares both uniform and masked models, showing consistent Gen. PPL improvements for AIPC.

7 Related Work

Hybrid AR-Diffusion Approaches SSD-LM (Han et al., 2023) pioneered block-wise generation but operates on a continuous probability simplex with higher computational cost; BlockGen uses discrete diffusion. BD3-LM (Arriola et al., 2025a) proposed single-block-size sequence modeling, enabling KV caching. BlockGen extends BD3-LM further by training on a mixture of block sizes. TiDAR (Liu et al., 2025a) adapts pre-trained AR models into masked multi-token generators with AR verification, akin to speculative decoding (Leviathan et al., 2023). We train from scratch and use AR predictions for predictor-corrector sampling rather than verification. Wu et al. (2025); Wang et al. (2025b) adapts pre-trained MDMs for block-wise caching. BlockGen instead proposes a pre-training objective to support block-by-block generation and KV caching natively. Huang & Tang (2025) use heuristics or RL to select block sizes at inference. BlockGen provides the pre-training foundation for adaptive block size selection. Eso-LMs (Sahoo et al., 2025b) define a mixture over partially masked sequences, and trains using a mixture of sequence-level MDM and AR training. Eso-LMs draft with diffusion, then fill autoregressively. We do the reverse (AR first, then diffusion once the KV cache grows). Eso-LMs are closer to any-order AR model (Uria et al., 2014; Strauss & Oliva, 2021; Hoogeboom et al., 2022; Shih et al., 2022; Kim et al., 2025a), compared to BlockGen that generates tokens left-to-right.

Predictor-Corrector Samplers Our AIPC sampler improves upon random correctors (Campbell et al., 2022; 2024; Gat et al., 2024; Wang et al., 2025a; Anonymous, 2026) by explicitly targeting inconsistent tokens rather than injecting noise at random positions. Unlike prior work that trains additional corrector modules (Lezama et al., 2023; Liu et al., 2025b; Zhao et al., 2025; Kim et al., 2025b; Zhang et al., 2025; Peng et al., 2025a;b), AIPC repurposes the autoregressive capabilities



(a) Uniform

(b) Masked

Figure 4: Comparison of AIPC with baseline PC samplers. AIPC achieves the best performance in both the masked and uniform settings.

of the base model without additional training. Most PC samplers also focus exclusively on masked diffusion, with Anonymous (2026) being a recent exception, though uninformed. AIPC supports both masked and uniform noise, using only the learned density from the base model. BlockGen enables alternative samplers without additional training.

8 Conclusion

We introduce BlockGen, a framework for blockwise sequence modeling over variable block sizes. BlockGen yields substantial improvements in density estimation over existing Block Diffusion Models. Variable-block-size training enables flexible sampling algorithms that improve text quality. When instantiated with USDMs, BlockGen generates higher-quality samples than masked Block Diffusion. AIPC improves quality by leveraging autoregressive predictions to correct inconsistencies, without additional training. Future work includes devising alternative samplers, fine-tuning to further boost AIPC, and applying BlockGen to other domains and component densities.

References

Anonymous. The diffusion duality, chapter II: ψ-samplers and efficient curriculum. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=RSIoYWIzaP.

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, 2025a. URL https://arxiv.org/abs/2503.09573.

Marianne Arriola, Yair Schiff, Hao Phung, Aaron Gokaslan, and Volodymyr Kuleshov. Encoder-decoder diffusion language models for efficient training and inference, 2025b. URL https://arxiv.org/abs/2510.22852.

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces, 2023. URL https://arxiv.org/abs/2107.03006.

Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. A neural probabilistic language model. In T. Leen, T. Dietterich, and V. Tresp (eds.), *Advances in Neural Information Processing Systems*, volume 13. MIT Press, 2000. URL https://proceedings.neurips.cc/paper_files/paper/2000/file/728f206c2a01bf572b5940d7d9a8fa4c-Paper.pdf.

Andrew Campbell, Joe Benton, Valentin De Bortoli, Tom Rainforth, George Deligiannidis, and Arnaud Doucet. A continuous time framework for discrete denoising models, 2022. URL https://arxiv.org/abs/2205.14987.

Andrew Campbell, Jason Yim, Regina Barzilay, Tom Rainforth, and Tommi Jaakkola. Generative flows on discrete state-spaces: Enabling multimodal flows with applications to protein co-design, 2024. URL https://arxiv.org/abs/2402.04997.

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. One billion word benchmark for measuring progress in statistical language modeling, 2014. URL https://arxiv.org/abs/1312.3005.

Justin Deschenaux and Caglar Gulcehre. Promises, outlooks and challenges of diffusion language modeling, 2024. URL https://arxiv.org/abs/2406.11473.

Sander Dieleman, Laurent Sartran, Arman Roshannai, Nikolay Savinov, Yaroslav Ganin, Pierre H. Richemond, Arnaud Doucet, Robin Strudel, Chris Dyer, Conor Durkan, Curtis Hawthorne, Rémi Leblond, Will Grathwohl, and Jonas Adler. Continuous diffusion for categorical data, 2022. URL https://arxiv.org/abs/2211.15089.

Juechu Dong, Boyuan Feng, Driss Guessous, Yanbo Liang, and Horace He. Flex attention: A programming model for generating optimized attention kernels, 2024. URL https://arxiv.org/abs/2412.05496.

Itai Gat, Tal Remez, Neta Shaul, Felix Kreuk, Ricky T. Q. Chen, Gabriel Synnaeve, Yossi Adi, and Yaron Lipman. Discrete flow matching, 2024. URL https://arxiv.org/abs/2407.15595.

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

Google. Gemma 3 technical report, 2025. URL https://arxiv.org/abs/2503.19786.

Xiaochuang Han, Sachin Kumar, and Yulia Tsvetkov. Ssd-lm: Semi-autoregressive simplex-based diffusion language model for text generation and modular control, 2023. URL https://arxiv.org/abs/2210.17432.

Zhengfu He, Tianxiang Sun, Kuanning Wang, Xuanjing Huang, and Xipeng Qiu. Diffusionbert: Improving generative masked language models with diffusion models, 2022. URL https://arxiv.org/abs/2211.15029.

Emiel Hoogeboom, Alexey A. Gritsenko, Jasmijn Bastings, Ben Poole, Rianne van den Berg, and Tim Salimans. Autoregressive diffusion models, 2022. URL https://arxiv.org/abs/2110.02037.

Chihan Huang and Hao Tang. Ctrldiff: Boosting large diffusion language models with dynamic block prediction and controllable generation, 2025. URL https://arxiv.org/abs/2505.14455.

Jaeyeon Kim, Lee Cheuk-Kit, Carles Domingo-Enrich, Yilun Du, Sham Kakade, Timothy Ngotiaoco, Sitan Chen, and Michael Albergo. Any-order flexible length masked diffusion, 2025a. URL https://arxiv.org/abs/2509.01025.

Jaeyeon Kim, Seunggeun Kim, Taekyun Lee, David Z. Pan, Hyeji Kim, Sham Kakade, and Sitan Chen. Fine-tuning masked diffusion for provable self-correction, 2025b. URL https://arxiv.org/abs/2510.01384.

Dirk P Kroese, Thomas Taimre, and Zdravko I Botev. *Handbook of Monte Carlo methods*. Wiley Series in Probability and Statistics. Wiley-Blackwell, Hoboken, NJ, February 2011.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding, 2023. URL https://arxiv.org/abs/2211.17192.

Jose Lezama, Tim Salimans, Lu Jiang, Huiwen Chang, Jonathan Ho, and Irfan Essa. Discrete predictor-corrector diffusion models for image synthesis. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=VM8batVBWvg.

Jingyu Liu, Xin Dong, Zhifan Ye, Rishabh Mehta, Yonggan Fu, Vartika Singh, Jan Kautz, Ce Zhang, and Pavlo Molchanov. Tidar: Think in diffusion, talk in autoregression, 2025a. URL https://arxiv.org/abs/2511.08923.

Sulin Liu, Juno Nam, Andrew Campbell, Hannes Stärk, Yilun Xu, Tommi Jaakkola, and Rafael Gómez-Bombarelli. Think while you generate: Discrete diffusion with planned denoising, 2025b. URL https://arxiv.org/abs/2410.06264.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution, 2024. URL https://arxiv.org/abs/2310.16834.

Meta. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Vaishnavh Nagarajan, Chen Henry Wu, Charles Ding, and Aditi Raghunathan. Roll the dice & look before you leap: Going beyond the creative limits of next-token prediction. *arXiv preprint arXiv:2504.15266*, 2025.

Hunter Nisonoff, Junhao Xiong, Stephan Allenspach, and Jennifer Listgarten. Unlocking guidance for discrete state-space diffusion and flow models. *arXiv preprint arXiv:2406.01572*, 2024.

OpenAI. Gpt-oss: open-weight language models by openai. https://github.com/openai/gpt-oss, 2024. GitHub repository.

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, 2025. URL https://arxiv.org/abs/2406.03736.

William Peebles and Saining Xie. Scalable diffusion models with transformers, 2023. URL https://arxiv.org/abs/2212.09748.

Fred Zhangzhi Peng, Zachary Bezemek, Sawan Patel, Jarrid Rector-Brooks, Sherwood Yao, Avishek Joey Bose, Alexander Tong, and Pranam Chatterjee. Path planning for masked diffusion model sampling, 2025a. URL https://arxiv.org/abs/2502.03540.

Fred Zhangzhi Peng, Zachary Bezemek, Jarrid Rector-Brooks, Shuibai Zhang, Anru R. Zhang, Michael Bronstein, Avishek Joey Bose, and Alexander Tong. Planner aware path learning in diffusion language models training, 2025b. URL https://arxiv.org/abs/2509.23405.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Anselm Levskaya, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference, 2022. URL https://arxiv.org/abs/2211.05102.

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

Subham Sekhar Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models, 2024. URL https://arxiv.org/abs/2406.07524.

Subham Sekhar Sahoo, Justin Deschenaux, Aaron Gokaslan, Guanghan Wang, Justin Chiu, and Volodymyr Kuleshov. The diffusion duality, 2025a. URL https://arxiv.org/abs/2506.10892.

Subham Sekhar Sahoo, Zhihan Yang, Yash Akhauri, Johnna Liu, Deepansha Singh, Zhoujun Cheng, Zhengzhong Liu, Eric Xing, John Thickstun, and Arash Vahdat. Esoteric language models, 2025b. URL https://arxiv.org/abs/2506.01928.

Yair Schiff, Subham Sekhar Sahoo, Hao Phung, Guanghan Wang, Sam Boshar, Hugo Dalla-torre, Bernardo P. de Almeida, Alexander Rush, Thomas Pierrot, and Volodymyr Kuleshov. Simple guidance mechanisms for discrete diffusion models, 2025. URL https://arxiv.org/abs/2412.10193.

Jiaxin Shi and Michalis K. Titsias. Demystifying diffusion objectives: Reweighted losses are better variational bounds, 2025. URL https://arxiv.org/abs/2511.19664.

Jiaxin Shi, Kehang Han, Zhe Wang, Arnaud Doucet, and Michalis K. Titsias. Simplified and generalized masked diffusion for discrete data, 2025. URL https://arxiv.org/abs/2406.04329.

Andy Shih, Dorsa Sadigh, and Stefano Ermon. Training and inference on any-order autoregressive models the right way, 2022. URL https://arxiv.org/abs/2205.13554.

Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics, 2015. URL https://arxiv.org/abs/1503.03585.

Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=PxTIG12RRHS.

Ryan R. Strauss and Junier B. Oliva. Arbitrary conditional distributions with energy, 2021. URL https://arxiv.org/abs/2102.04426.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2023. URL https://arxiv.org/abs/2104.09864.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023. URL https://arxiv.org/abs/2307.09288.

Benigno Uria, Iain Murray, and Hugo Larochelle. A deep and tractable density estimator. In Eric P. Xing and Tony Jebara (eds.), *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pp. 467–475, Bejing, China, 22–24 Jun 2014. PMLR. URL https://proceedings.mlr.press/v32/uria14.html.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023. URL https://arxiv.org/abs/1706.03762.

Guanghan Wang, Yair Schiff, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Remasking discrete diffusion models with inference-time scaling, 2025a. URL https://arxiv.org/abs/2503.00307.

Xu Wang, Chenkai Xu, Yijie Jin, Jiachun Jin, Hao Zhang, and Zhijie Deng. Diffusion llms can do faster-than-ar inference via discrete diffusion forcing, 2025b. URL https://arxiv.org/abs/2508.09192.

Chengyue Wu, Hao Zhang, Shuchen Xue, Shizhe Diao, Yonggan Fu, Zhijian Liu, Pavlo Molchanov, Ping Luo, Song Han, and Enze Xie. Fast-dllm v2: Efficient block-diffusion llm, 2025. URL https://arxiv.org/abs/2509.26328.

Shuibai Zhang, Fred Zhangzhi Peng, Yiheng Zhang, Jin Pan, and Grigorios G. Chrysos. Corrective diffusion language models, 2025. URL https://arxiv.org/abs/2512.15596.

Yixiu Zhao, Jiaxin Shi, Feng Chen, Shaul Druckmann, Lester Mackey, and Scott Linderman. Informed correctors for discrete diffusion models, 2025. URL https://arxiv.org/abs/2407.21243.

Table 2: **Throughput and Generative Perplexity (Gen. PPL) as a function of block size L' and AR prefix length k.** We report throughput (tok/s) and generative perplexity under GPT-2 Large (Gen. PPL; ↓), with unigram entropy H_1 in parentheses (↑). Standard block diffusion (BDM) uses no AR prefix (k=0). BlockGen with AIPC matches BDM throughput while improving Gen. PPL, and is less sensitive to shorter prefixes due to AR-guided re-injection. Full AIPC results across block sizes, prefix lengths, and baselines are in Suppl. 4.

Variant	Block size L'	$k=0$		$k=128$		$k=256$	
		Gen. PPL (H1)	Thr. (tok/s)	Gen. PPL (H1)	Thr. (tok/s)	Gen. PPL (H1)	Thr. (tok/s)
BlockGen							
Uniform (AIPC; ours)	16	–	–	**52.4** (5.3)	1740	**55.8** (5.4)	1735
Masked (AIPC; ours)	16	–	–	88.3 (5.5)	1874	78.1 (5.5)	1842
Uniform (AIPC; ours)	32	–	–	56.7 (5.3)	2811	58.5 (5.4)	2672
Uniform (AIPC; ours)	64	–	–	56.7 (5.3)	3471	57.0 (5.3)	3614
BDM							
Uniform (BlockDiff)	16	116.7 (5.5)	1579	–	–	–	–
Masked (BlockDiff)	16	280.4 (5.7)	1737	–	–	–	–

Table 3: Validation Perplexlity (Val. PPL) on OWT after 1M training steps. We train on block sizes $\{2^i\}_{i=0}^{K}$, such that $2^K = L_{\max}$ with varying $\boldsymbol{\gamma}$ (Sec. 5.2). **Takeaway:** "Mostly AR" weights achieve near-AR perplexity (17.3-17.4). See Table 6 for the ELBO for each (single) block size.

Model	Val. PPL		
	$L_{\max}$=16	$L_{\max}$=32	$L_{\max}$=64
Uniform Weights			
M-BlockGen (ours)	17.5	–	–
U-BlockGen (ours)	18.5	18.9	19.3
Mostly AR Weights			
U-BlockGen (ours)	17.3	17.4	17.4

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, 2025. URL https://arxiv.org/abs/2409.02908.

A Additional Experimental Details

A.1 Data Preparation

For the One Billion Words (LM1B) dataset, we follow the preprocessing of Lou et al. (2024); Sahoo et al. (2024)[1] and tokenize with `bert-base-uncased` following He et al. (2022). We use both wrapped and non-wrapped variants following Sahoo et al. (2025a); Arriola et al. (2025a). For OpenWebText, we tokenize with `GPT2`, concatenate sequences to length 1024 with `eos` tokens between them, and reserve the last 100k documents for validation.

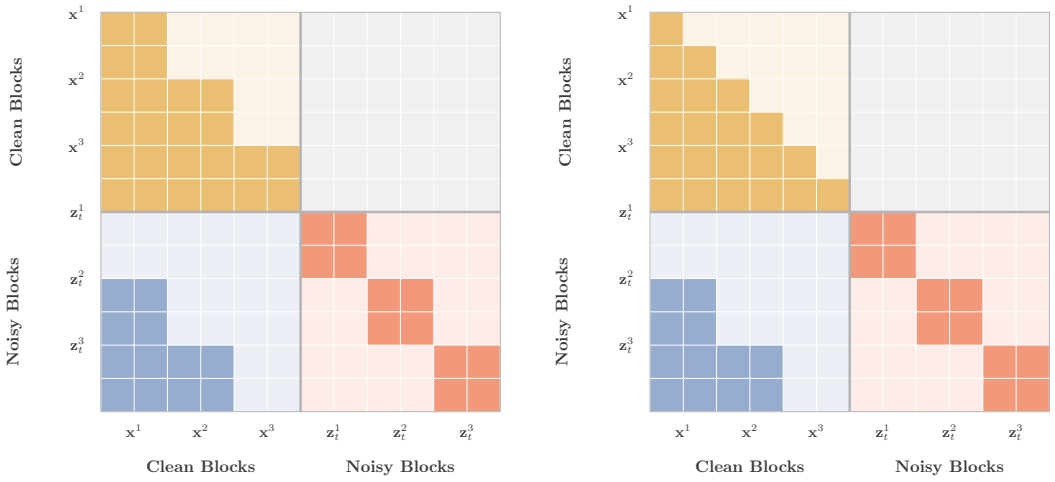
A.2 Denoising Backbone

We parameterize all models using the modified diffusion transformer architecture (Peebles & Xie, 2023) from Lou et al. (2024); Sahoo et al. (2024). We use 12 layers, a hidden dimension of 768, 12 attention heads, and a timestep embedding dimension of 128. Following prior work (Sahoo et al., 2024; Schiff et al., 2025; Arriola et al., 2025a; Sahoo et al., 2025a), we keep the adaptive layernorm but feed it a zero vector, making the denoiser time-unconditional. This contrasts with prior work on USDMs (Schiff et al., 2025; Sahoo et al., 2025a), where time-conditioning is used to improves performance. We chose this approach for simplicity, to avoid modifying the backbone. In principle,

[1] https://github.com/louaaron/Score-Entropy-Discrete-Diffusion/blob/main/data.py

Algorithm 2: Stratified Block Size Selection

Require: Weights $\boldsymbol{\gamma} \in \Delta^M$, number of GPUs D
$c_0 \leftarrow 0, \quad c_n \leftarrow \sum_{j=1}^{n} \gamma_j$ for $n = 1, \ldots, M$
$u \sim \text{Uniform}(0, 1)$
for $d = 1$ **to** D **do**
  $\tilde{u}_d \leftarrow \left(\frac{d-1}{D} + u\right) \mod 1$
  $i_d \leftarrow n$ such that $c_{n-1} \leq \tilde{u}_d < c_n$
end for
return $(i_1, \ldots, i_D)$



(a) Standard block diffusion attention.

(b) Block Diffusion with Causal attention on the clean prefix.

Figure 5: **Attention patterns for block diffusion (training)** with $L' = 2$. Noisy blocks attend to all tokens within the block and to all clean tokens in previous blocks. (a) shows the attention from Arriola et al. (2025a). (b) uses causal attention over the clean tokens. This is important for BlockGen since we train over multiple block sizes but want to share a single KV cache across all block sizes during inference. Unlike Arriola et al. (2025a), we concatenate the clean context before the noisy tokens. This preserves correctness and does not affect performance, with a more intuitive pattern where tokens attend only to previous tokens in the input.

one could enable both time-conditioning and KV-caching by using a zero time embedding for clean tokens (so their activations remain cacheable during sampling) and use the actual time embedding for noisy blocks only. However, this introduces differences in the denoiser compared to prior work, therefore, we keep the denoiser time-unconditional in this work.

A.3 Attention Implementation

Figure 5 shows the attention pattern used during training. Following Arriola et al. (2025a), we feed the denoising network sequences of 2x the context length (256 on LM1B, 2048 on OWT) to obtain predictions for all noisy blocks conditioned on the clean prefix. We use a slightly modified version of their FlexAttention (Dong et al., 2024) attention masks. While Arriola et al. (2025a) concatenate sequences with clean tokens second, we place the clean context first and noisy tokens second. This ordering does not affect speed or correctness but yields a more natural attention pattern that mirrors standard AR models, where tokens attend to the past rather than the future.

A.4 Optimization hyperparameters

We use the Adam optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$, batch size 512, learning rate $3e-4$ with linear warmup over 2500 steps, no cooldown, and no weight decay. We train in mixed precision with gradient clipping to 1 for 1M steps. We maintain an exponential moving average of the weights with decay 0.9999 and use a dropout rate of 0.1. We do not tune hyperparameters further, following prior work and to save compute.

A.5 Evaluating the Sample Quality

Generative perplexity (Gen. PPL). We evaluate the quality of generated text using the perplexity under a larger reference language model (GPT-2 Large), following prior work (Lou et al., 2024; Sahoo et al., 2024; 2025a). Given N generated sequences $\{\mathbf{x}^{(i)}\}_{i=1}^{N}$ (each tokenized with the GPT-2 tokenizer and of length L), we compute

$$\text{Gen. PPL} \;=\; \exp\left(-\frac{1}{NL}\sum_{i=1}^{N}\sum_{t=1}^{L}\log p_{\text{GPT-2 Large}}\left(x_t^{(i)} \mid x_{<t}^{(i)}\right)\right). \tag{10}$$

Unigram entropy. Since a low Gen. PPL can be achieved by degenerate repetitive text, we also report the average unigram entropy of generated samples (Dieleman et al., 2022). Let $\mathcal{V}$ be the GPT-2 vocabulary and $c(v, \mathbf{x}^{(i)})$ the number of occurrences of token $v \in \mathcal{V}$ in sequence $\mathbf{x}^{(i)}$. We define the empirical unigram distribution $q^{(i)}(v) = c(v, \mathbf{x}^{(i)})/L$ and report

$$\text{Entropy} \;=\; -\frac{1}{N}\sum_{i=1}^{N}\sum_{v\in\mathcal{V}} q^{(i)}(v)\,\log q^{(i)}(v). \tag{11}$$

B Additional Experimental Results

B.1 Complete AIPC Results and Baselines

We provide comprehensive AIPC results across different block sizes, AR prefix lengths, and noise types (uniform vs masked). Table 4 shows the organization of all detailed tables, which appear at the end of this appendix. In the result tables, we highlight in blue the hyperparameters whose associated results are plotted in the main body.

Table 4: Navigation table for detailed AIPC results. Each cell references the corresponding table with full experimental details.

AR Prefix	Block Size 16		Block Size 32		Block Size 64	
	Uniform	Masked	Uniform	Masked	Uniform	Masked
64	12	15	–	–	–	–
128	11	14	17	–	19	–
256	10	13	16	–	18	–

Predictor-Corrector Baselines We compare AIPC against two predictor-corrector baselines that replace the token selector $g(\cdot)$ in Algo. 1 with simpler alternatives requiring no AR predictions. The first baseline selects tokens uniformly at random (Algo. 3). The second baseline selects tokens with highest entropy under the diffusion marginals p^{D} (Algo. 4). As explained below, we do not use the Entropy-based selection with MDMs.

Entropy-Based Selection for Masked Diffusion Entropy is not a reliable remasking criterion for masked diffusion models. At unmasked positions with clean inputs, the model directly observes the token value. Moreover, these positions receive no training signal during pretraining, so their entropy values are not calibrated to reflect token quality. This makes entropy unsuitable for identifying low-quality tokens in MDMs, hence we do not include it as a baseline for masked models.

Algorithm 3 Uniform Random Re-injection Mask (baseline)

Require: Block length L', num. positions k
1: Sample $\mathcal{I} \subseteq \{1, \dots, L'\}$ uniformly with $|\mathcal{I}| = k$
2: $\mathbf{m}_\ell \leftarrow \mathbb{1}[\ell \in \mathcal{I}]$ for $\ell = 1, \dots, L'$
3: **return** $\mathbf{m}$

Algorithm 4 Diffusion-Metric Re-injection Mask (baseline)

Require: Diffusion marginals $p^{\mathrm{D}} \in \Delta^{L' \times |\mathcal{V}|}$, metric $\mu \in \{\textsc{Entropy}, \textsc{Confidence}, \textsc{Margin}\}$, num. positions k
1: $\mathbf{s} \leftarrow \textsc{DiffMetricScore}(p^{\mathrm{D}}, \mu)$ ▷ Higher = more uncertain (see text)
2: $\mathcal{I} \leftarrow$ indices of the top-k values in $\{s_\ell\}_{\ell=1}^{L'}$
3: $\mathbf{m}_\ell \leftarrow \mathbb{1}[\ell \in \mathcal{I}]$ for $\ell = 1, \dots, L'$
4: **return** $\mathbf{m}$

C Computing the Validation Perplexity

This section defines how we compute the validation perplexity.

Token- and sequence-level evaluation Let the validation set contain J sequences, indexed by $j \in \{1, \dots, J\}$. Each sequence is represented as $\mathbf{x}_{(j)} \in \mathcal{V}^L$ with an associated binary mask $\mathbf{a}_{(j)} \in \{0, 1\}^L$, where $a_{(j),\ell} = 1$ indicates that position ℓ is not padding. We evaluate a per-token loss vector $\mathcal{L}(\mathbf{x}_{(j)}) \in \mathbb{R}^L$ and compute the sequence-level NLL over non-padding tokens:

$$\widehat{\mathrm{NLL}}_{(j)} := \sum_{\ell=1}^{L} a_{(j),\ell}\, \mathcal{L}(\mathbf{x}_{(j)})_\ell. \tag{12}$$

For diffusion models, $\widehat{\mathrm{NLL}}_{(j)}$ is computed via Monte Carlo, as described in Suppl. C.1 and Suppl. C.2.

Dataset-level evaluation Let $N_{(j)} := \sum_{\ell=1}^{L} a_{(j),\ell}$ denote the number of non-padding tokens in sequence j, and let $N_{\mathrm{val}} := \sum_{j=1}^{J} N_{(j)}$. We compute the dataset-level perplexity as

$$\widehat{\mathrm{NLL}}_{\mathrm{val}} := \sum_{j=1}^{J} \widehat{\mathrm{NLL}}_{(j)}, \qquad \mathrm{PPL} := \exp\left(\frac{\widehat{\mathrm{NLL}}_{\mathrm{val}}}{N_{\mathrm{val}}}\right). \tag{13}$$

Thus, our reported perplexity is $\exp(\widehat{\mathrm{NLL}}_{\mathrm{val}}/N_{\mathrm{val}})$. The next paragraphs specify $\mathcal{L}$ for each model class.

C.1 Computing the NLL for pure models

This subsection specifies $\mathcal{L}$ when using a single factorization (AR, MDM, or Duo).

Autoregressive models For AR models, we use the exact token-level negative log-likelihood

$$\mathcal{L}_{\mathrm{AR}}(\mathbf{x}_{(j)})_\ell := -\log p_\theta(x_{(j),\ell} \mid x_{(j),<\ell}), \tag{14}$$

and set $\mathcal{L}(\mathbf{x}_{(j)}) := \mathcal{L}_{\mathrm{AR}}(\mathbf{x}_{(j)})$ in (12).

Masked diffusion (MDM) For masked diffusion (Sahoo et al., 2024; Shi et al., 2025; Ou et al., 2025), the negative ELBO (NELBO) yields an NLL bound of the form

$$\mathrm{NELBO}_{\mathrm{MDM}}(\mathbf{x}) := -\mathbb{E}_{t \sim \mathcal{U}[0,1],\, \mathbf{z}_t \sim q_t(\cdot \mid \mathbf{x}; \alpha_t)} \left[\frac{\alpha_t'}{1 - \alpha_t} \sum_{\ell=1}^{L} \log \langle \hat{\mathbf{x}}_\theta^\ell(\mathbf{z}_t, t), \mathbf{x}^\ell \rangle \right] \approx -\frac{\alpha_{\tilde{t}}'}{1 - \alpha_{\tilde{t}}} \sum_{\ell=1}^{L} \log \langle \hat{\mathbf{x}}_\theta^\ell(\mathbf{z}_{\tilde{t}}, \tilde{t}), \mathbf{x}^\ell \rangle, \tag{15}$$

which corresponds to a weighted cross-entropy loss over masked positions. $\tilde{t} \sim \mathcal{U}[0, 1]$ and $\mathbf{z}_{\tilde{t}} \sim q_{\tilde{t}}(\cdot \mid \mathbf{x}; \alpha_{\tilde{t}})$. Consistent with Sahoo et al. (2024), we use a single sample per sequence.

Uniform-state diffusion (Duo) For uniform-state diffusion (Schiff et al., 2025; Sahoo et al., 2025a), the NELBO reads as

$$\mathrm{NELBO}_{\mathrm{USDM}}(\mathbf{x}) := \mathbb{E}_{t\sim\mathcal{U}[0,1],\ \mathbf{z}_t\sim q_t(\cdot|\mathbf{x};\alpha_t)}\left[\sum_{\ell=1}^{L} f_{\mathrm{ELBO}}\big(\mathbf{z}_t^\ell, \hat{\mathbf{x}}_\theta^\ell(\mathbf{z}_t,t),\alpha_t;\mathbf{x}^\ell\big)\right] \approx \sum_{\ell=1}^{L} f_{\mathrm{ELBO}}\big(\mathbf{z}_{\tilde{t}}^\ell, \hat{\mathbf{x}}_\theta^\ell(\mathbf{z}_{\tilde{t}},\tilde{t}),\alpha_{\tilde{t}};\mathbf{x}^\ell\big), \tag{16}$$

with per-token term

$$f_{\mathrm{ELBO}}(\mathbf{z}_t,\hat{\mathbf{x}},\alpha_t;\mathbf{x}) := -\frac{\alpha_t'}{|\mathcal{V}|\alpha_t}\left[\frac{|\mathcal{V}|}{\bar{\mathbf{x}}_i} - \frac{|\mathcal{V}|}{(\bar{\mathbf{x}}_\theta)_i} - \sum_r \frac{\bar{\mathbf{x}}_r}{\bar{\mathbf{x}}_i}\log\frac{(\bar{\mathbf{x}}_\theta)_i\,\bar{\mathbf{x}}_r}{(\bar{\mathbf{x}}_\theta)_r\,\bar{\mathbf{x}}_i}\right], \tag{17}$$

where $\bar{\mathbf{x}} = |\mathcal{V}|\alpha_t\mathbf{x} + (1-\alpha_t)\mathbf{1}$, $\bar{\mathbf{x}}_\theta = |\mathcal{V}|\alpha_t\hat{\mathbf{x}} + (1-\alpha_t)\mathbf{1}$, and $i = \arg\max_r(\mathbf{z}_t)_r$. Recall that $|\mathcal{V}|$ denotes the vocabulary size. $\tilde{t}\sim\mathcal{U}[0,1]$ and $\mathbf{z}_{\tilde{t}}\sim q_{\tilde{t}}(\cdot\mid\mathbf{x};\alpha_{\tilde{t}})$. Following Sahoo et al. (2025a), we use a single Monte Carlo estimate per sequence.

C.2 Computing the ELBO with BlockGen

ELBO computation for a single block size Fix a block size s and a sequence $\mathbf{x}_{(j)}$. For each Monte Carlo draw, we sample $t\sim\mathcal{U}[0,1]$. We then sample $\mathbf{z}_t\sim q_t(\cdot\mid\mathbf{x}_{(j)};\alpha_t)$. Using the denoiser output and the chosen objective, we compute per-token losses $\mathcal{L}_s(\mathbf{x}_{(j)};t,\mathbf{z}_t)\in\mathbb{R}^L$. The corresponding sequence-level NLL is

$$\mathrm{NLL}_{s,(j)}(t,\mathbf{z}_t) := \sum_{\ell=1}^{L} a_{(j),\ell}\,\mathcal{L}_s(\mathbf{x}_{(j)};t,\mathbf{z}_t)_\ell. \tag{18}$$

We estimate the expected sequence NLL with K_{MC}=8 Monte Carlo draws:

$$\widehat{\mathrm{NLL}}_{s,(j)} := \frac{1}{K_{\mathrm{MC}}}\sum_{k=1}^{K_{\mathrm{MC}}}\mathrm{NLL}_{s,(j)}(t_k,\mathbf{z}_{t_k}). \tag{19}$$

We use K_{MC}=8, and found it stable accross seeds.

Evaluation of the BlockGen ELBO For the BlockGen parameterization (6) with block sizes $\{s_i\}_{i=1}^M$ and weights $\boldsymbol{\gamma}$, we first compute $\widehat{\mathrm{NLL}}_{s_i,(j)}$ for each i with $\gamma_i>0$ using (19). For evaluation, we always report the log-sum-exp (LSE) bound

$$\widehat{\mathrm{NLL}}_{\mathrm{LSE},(j)} := -\log\sum_{i:\gamma_i>0}\gamma_i\exp\Big(-\widehat{\mathrm{NLL}}_{s_i,(j)}\Big), \tag{20}$$

and set $\widehat{\mathrm{NLL}}_{(j)} := \widehat{\mathrm{NLL}}_{\mathrm{LSE},(j)}$ in (13). We finally average over the dataset and exponentiate to obtain the final ELBO.

D Proofs

D.1 BlockGen ELBOs

Recall the definition of the BlockGen mixture density from (6):

$$p_\theta^{\mathrm{BlockGen}}(\mathbf{x}) = \sum_{i=1}^{M}\gamma_i\,p_\theta^{(s_i)}(\mathbf{x}), \tag{21}$$

where $\boldsymbol{\gamma}\in\Delta^M$ represents the mixture weights and each $p_\theta^{(s_i)}$ is a valid density. We assume each component i admits a tractable lower bound $\mathcal{E}^{(s_i)}(\theta,\mathbf{x})$ such that:

$$\log p_\theta^{(s_i)}(\mathbf{x}) \geq \mathcal{E}^{(s_i)}(\theta,\mathbf{x}) \implies p_\theta^{(s_i)}(\mathbf{x}) \geq e^{\mathcal{E}^{(s_i)}(\theta,\mathbf{x})}. \tag{22}$$

Substituting this inequality into the mixture definition:

$$p_\theta^{\mathrm{BlockGen}}(\mathbf{x}) = \sum_{i=1}^{M}\gamma_i\,p_\theta^{(s_i)}(\mathbf{x}) \geq \sum_{i=1}^{M}\gamma_i\,e^{\mathcal{E}^{(s_i)}(\theta,\mathbf{x})}. \tag{23}$$

Taking the logarithm of both sides yields the *log-sum-exp bound*:

$$\log p_\theta^{\text{BlockGen}}(\mathbf{x}) \geq \log \left(\sum_{i=1}^{M} \gamma_i \, e^{\mathcal{E}^{(s_i)}(\theta, \mathbf{x})} \right) = \log \left(\sum_{i=1}^{M} e^{\mathcal{E}^{(s_i)}(\theta, \mathbf{x}) + \log \gamma_i} \right). \tag{24}$$

Finally, we can derive the *mixture likelihood bound* using Jensen's inequality and the concavity of the log:

$$\log \left(\sum_{i=1}^{M} \gamma_i \, e^{\mathcal{E}^{(s_i)}(\theta, \mathbf{x})} \right) \geq \sum_{i=1}^{M} \gamma_i \log \left(e^{\mathcal{E}^{(s_i)}(\theta, \mathbf{x})} \right) = \sum_{i=1}^{M} \gamma_i \, \mathcal{E}^{(s_i)}(\theta, \mathbf{x}). \tag{25}$$

D.2 Alternative Parameterization via Geometric Mean

We show that parameterizing the generative model with a geometric mean of component densities, denoted as p_θ^{GMP} also admits the mixture likelihood bound:

$$p_\theta^{\text{GMP}}(\mathbf{x}) = \frac{1}{Z_\theta} \prod_{i=1}^{M} \left(p_\theta^{(s_i)}(\mathbf{x}) \right)^{\gamma_i}, \quad \text{where} \quad Z_\theta = \sum_{\mathbf{x}' \in \mathcal{V}^L} \prod_{i=1}^{M} \left(p_\theta^{(s_i)}(\mathbf{x}') \right)^{\gamma_i}. \tag{26}$$

The log-likelihood is given by:

$$\log p_\theta^{\text{GMP}}(\mathbf{x}) = \sum_{i=1}^{M} \gamma_i \log p_\theta^{(s_i)}(\mathbf{x}) - \log Z_\theta. \tag{27}$$

To lower-bound (27), we upper-bound the partition function Z_θ. For $p, q \in [1, \infty]$ with $1/p + 1/q = 1$, Hölder's inequality reads

$$\sum_{\mathbf{x}'} |f(\mathbf{x}') g(\mathbf{x}')| \leq \left(\sum_{\mathbf{x}'} |f(\mathbf{x}')|^p \right)^{1/p} \left(\sum_{\mathbf{x}'} |g(\mathbf{x}')|^q \right)^{1/q}. \tag{28}$$

Applying this inequality repeatedly yields the generalized form: for $p_i \in [1, \infty]$ with $\sum_i 1/p_i = 1$,

$$\sum_{\mathbf{x}'} \prod_{i=1}^{M} |f_i(\mathbf{x}')| \leq \prod_{i=1}^{M} \left(\sum_{\mathbf{x}'} |f_i(\mathbf{x}')|^{p_i} \right)^{1/p_i}. \tag{29}$$

Assume $\gamma_i > 0$ (terms with $\gamma_i = 0$ equal 1 and can be ignored), set $f_i(\mathbf{x}') := (p_\theta^{(s_i)}(\mathbf{x}'))^{\gamma_i}$ and choose $p_i := 1/\gamma_i$ so that $\sum_i 1/p_i = \sum_i \gamma_i = 1$. Applying the generalized form to Z_θ gives:

$$Z_\theta = \sum_{\mathbf{x}'} \prod_{i=1}^{M} (p_\theta^{(s_i)}(\mathbf{x}'))^{\gamma_i} \leq \prod_{i=1}^{M} \left(\sum_{\mathbf{x}'} \left((p_\theta^{(s_i)}(\mathbf{x}'))^{\gamma_i} \right)^{1/\gamma_i} \right)^{\gamma_i} = \prod_{i=1}^{M} \left(\sum_{\mathbf{x}'} p_\theta^{(s_i)}(\mathbf{x}') \right)^{\gamma_i}. \tag{30}$$

Since each component $p_\theta^{(s_i)}$ is normalized, $\sum_{\mathbf{x}'} p_\theta^{(s_i)}(\mathbf{x}') = 1$. Therefore:

$$Z_\theta \leq \prod_{i=1}^{M} (1)^{\gamma_i} = 1 \implies \log Z_\theta \leq 0 \implies -\log Z_\theta \geq 0. \tag{31}$$

Substituting this back into the log-likelihood:

$$\log p_\theta^{\text{GMP}}(\mathbf{x}) \geq \sum_{i=1}^{M} \gamma_i \log p_\theta^{(s_i)}(\mathbf{x}). \tag{32}$$

Finally, using the per-component tractable lower bound assumption $\log p_\theta^{(s_i)}(\mathbf{x}) \geq \mathcal{E}^{(s_i)}(\theta, \mathbf{x})$:

$$\log p_\theta^{\text{GMP}}(\mathbf{x}) \geq \sum_{i=1}^{M} \gamma_i \, \mathcal{E}^{(s_i)}(\theta, \mathbf{x}). \tag{33}$$

Thus, the Geometric Mean Parameterization also admits the *Jensen bound* as a valid ELBO.

Table 5: Validation perplexity after 250k training steps for single-block-size models. Masked GBD trained with cross-entropy (CE) matches or outperforms block diffusion trained with the ELBO (Shi & Titsias, 2025) and performs comparably to Arriola et al. (2025a) without their variance-reduction scheme. With uniform noise, CE improves perplexity only at small context sizes. LM1B (wrap) uses sentence packing; LM1B (no wrap) pads shorter sequences. All models are trained from scratch.

	LM1B (wrap)				LM1B (no wrap)				OWT					
	L=1	L=4	L=16	L=128	L=1	L=4	L=16	L=128	L=1	L=4	L=16	L=32	L=128	L=1024
Baselines														
AR	22.4	–	–	–	20.5	–	–	–	17.0	–	–	–	–	–
MDLM (Sahoo et al., 2024)	–	–	–	33.7	–	–	–	33.0	–	–	–	–	–	24.9
Duo (Sahoo et al., 2025a)	–	–	–	39.2	–	–	–	38.3	–	–	–	–	–	27.5
Masked Block Diffusion														
BD (Arriola et al., 2025a)	22.4	28.9	32.6	–	20.5	30.3	32.3	–	17.0	20.9	22.8	–	–	–
GBD (ELBO; ours)	–	30.1	33.6	–	–	31.2	34.5	–	–	21.8	23.7	24.1	24.5	–
GBD (CE; ours)	22.4	28.2	31.9	–	20.5	27.3	31.0	–	17.0	20.9	22.8	23.5	24.2	–
Uniform Block Diffusion														
GBD (ELBO; ours)	–	33.7	38.2	–	–	34.4	38.3	–	–	23.3	25.7	25.9	27.1	–
GBD (CE; ours)	22.7	31.9	36.9	–	21.2	30.2	35.3	–	17.0	22.9	26.6	28.4	28.7	–

Table 6: Validation perplexity for BlockGen after **250k steps** of training on OWT. "ELBO" denotes the log-sum-exp bound in (7). In addition, we report the single-block size performance. A clear trade-off appears between AR and large-block size performance. Adjusting the fraction of training steps per block size does not improve both metrics simultaneously. Stratified block size selection (Sec. 3.2) always improves performance.

Weights per block size			Stratified	ELBO	Single Block Size ELBO							
1	128	others		(LSE)	L=1	L=2	L=4	L=8	L=16	L=32	L=64	L=128
Masked Diffusion												
0.125	0.125	0.125	×	19.3	19.3	20.3	21.6	22.8	23.7	24.5	25.2	25.8
0.5	0.5	0.0	×	17.9	17.9	20.7	23.1	24.6	25.6	26.3	26.8	27.1
Uniform Diffusion												
0.125	0.125	0.125	×	21.2	21.2	23.1	24.5	**25.5**	**26.3**	**27.0**	**27.6**	**28.3**
0.5	0.5	0.0	×	19.7	19.7	36.1	39.4	37.5	34.2	30.5	28.9	28.6
0.5	0.1	$0.0\bar{6}$	×	19.4	19.4	21.7	24.1	26.0	27.4	28.5	29.4	30.2
0.6	0.1	0.05	×	18.8	18.8	22.0	24.7	26.6	28.1	29.2	30.1	30.9
0.7	0.05	$0.041\bar{6}$	×	18.5	18.5	21.9	25.1	27.4	29.2	30.5	31.6	32.6
0.7	0.05	$0.041\bar{6}$	✓	18.5	18.5	21.4	**24.4**	26.8	28.6	30.0	31.1	32.1
0.89	0.05	0.01	×	17.8	17.8	23.4	28.1	31.3	33.6	35.3	36.5	37.7
0.89	0.05	0.01	✓	**17.6**	**17.6**	22.0	26.3	29.6	32.0	33.7	35.0	36.2
0.7	0.15	0.025	✓	18.4	18.4	**21.6**	24.5	26.7	28.2	29.4	30.2	31.0
0.79	0.15	0.01	✓	18.1	18.1	23.1	26.6	28.8	30.2	31.2	31.9	32.6

Table 7: Validation perplexity BlockGen after **1M steps** of training on OWT. "ELBO" denotes the log-sum-exp bound in (7). We also report per-block-size ELBOs, which estimate performance when sampling with a fixed block size. The trade-off between AR performance and large-block performance persists at 1M steps and mirrors Table 6. With the "Mostly AR" choice of γ, training with a larger maximum block size can improve intermediate block sizes without hurting the AR component. For example, max 64 improves performance at block size 32 relative to training only up to 32. A similar pattern holds for max 32 versus 16. We use stratified block-size selection (Sec. 3.2) in all experiments.

	Weights			ELBO	Single Block-Size ELBO						
Model	L'=1	L'_{max}	others	(LSE)	L'=1	L'=2	L'=4	L'=8	L'=16	L'=32	L'=64
Uniform Weights											
Masked BlockGen (1-16)	0.2	0.2	0.2	17.5	17.5	18.6	20.3	22.1	23.7	–	–
Uniform BlockGen (1-16)	0.2	0.2	0.2	18.5	18.5	19.8	21.6	23.4	25.1	–	–
Uniform BlockGen (1-32)	$0.1\bar{6}$	$0.1\bar{6}$	$0.1\bar{6}$	18.9	18.9	20.2	21.8	23.4	24.7	25.8	–
Uniform BlockGen(1-64)	0.142	0.142	0.142	19.3	19.3	20.5	22.1	23.4	24.5	25.4	26.2
Mostly AR Weights											
Uniform BlockGen (1-16)	0.7	0.15	0.05	17.3	17.3	20.2	23.0	25.5	27.5	–	–
Uniform BlockGen (1-32)	0.7	0.15	0.0375	17.4	17.4	20.3	23.2	25.4	27.1	28.5	–
Uniform BlockGen (1-64)	0.7	0.15	0.03	17.4	17.4	20.5	23.3	25.4	26.9	28.1	29.0

Table 8: Generative Perplexity (Gen. PPL, evaluated by GPT-2 Large) and throughput over 1024 samples. **Takeaway:** Block diffusion trades quality for speed. Uniform BDMs (86.5 PPL at 64 steps/block) significantly outperform Masked BDMs (146.0 PPL) due to self-correction. KV-caching enables significantly higher throughput than full-sequence diffusion. Throughput measured at batch size 32.

Model	Gen. PPL ↓	Entropy ↑	Latency ↓ (ms)	Throughput ↑ (tok/s)
AR				
1024 steps	39.6	5.5	21.7 ± 1.5	589.27 ± 22
MDLM				
32 steps	192.3	5.7	0.6 ± 0.0	$4\,077.1 \pm 3.1$
64 steps	142.6	5.7	1.2 ± 0.0	$2\,070.7 \pm 0.7$
128 steps	122.9	5.7	2.5 ± 0.0	$1\,043.2 \pm 0.2$
256 steps	114.0	5.6	5.9 ± 0.1	523.90 ± 0.06
512 steps	109.1	5.6	12.0 ± 0.1	262.26 ± 0.33
1024 steps	106.7	5.6	23.9 ± 0.9	131.42 ± 0.05
Duo				
32 steps	98.3	5.6	0.5 ± 0.0	$7\,465.6 \pm 3.1$
64 steps	87.3	5.6	1.1 ± 0.0	$3\,831.9 \pm 1.2$
128 steps	81.3	5.5	2.1 ± 0.0	$1\,940.8 \pm 0.2$
256 steps	79.2	5.5	4.3 ± 0.0	977.8 ± 0.6
512 steps	78.1	5.5	8.3 ± 0.1	490.5 ± 0.1
1024 steps	77.7	5.5	18.5 ± 1.3	244.6 ± 1.0
Masked Block Diffusion ($L' = 16$)				
4 s/b (256)	1262.6	5.9	3.9 ± 0.0	$6\,754.9 \pm 10.2$
8 s/b (512)	587.6	5.8	7.9 ± 0.0	3320.4 ± 9.6
16 s/b (1024)	280.4	5.7	14.8 ± 0.1	$1\,737.1 \pm 5.0$
32 s/b (2048)	187.8	5.7	30.4 ± 2.6	848.9 ± 2.6
64 s/b (4096)	146.0	5.7	59.6 ± 0.1	432.9 ± 1.0
Uniform Block Diffusion ($L' = 16$)				
4 s/b (256)	661.3	5.5	3.9 ± 0.0	6192.7 ± 14.3
8 s/b (512)	211.8	5.5	7.7 ± 0.2	3138.0 ± 11.3
16 s/b (1024)	116.7	5.5	15.2 ± 0.0	1579.5 ± 6.4
32 s/b (2048)	95.5	5.5	30.2 ± 0.2	791.3 ± 6.6
64 s/b (4096)	86.5	5.5	59.6 ± 0.4	396.8 ± 1.4

Table 9: Hybrid sampling: generate k tokens with AR, then continue with block diffusion. Models trained for 1M steps on OWT. **Takeaway:** Longer AR prefixes substantially improve Gen. PPL. With k=256, Uniform BlockGen reaches 98.0 PPL, closing the gap to full-sequence Duo (77.7 PPL in Table 8).

	Prefix $k = 64$		Prefix $k = 128$		Prefix $k = 256$	
Model	Gen. PPL ↓	Entropy ↑	Gen. PPL ↓	Entropy ↑	Gen. PPL ↓	Entropy ↑
Masked BlockGen ($L' \in \{1, 2, 4, 8, 16\}$, uniform weights)						
4 s/b (256)	675.7	5.7	545.9	5.7	357.2	5.7
8 s/b (512)	437.7	5.8	356.3	5.7	259.4	5.7
16 s/b (1024)	288.9	5.7	265.4	5.7	177.7	5.7
32 s/b (2048)	242.2	5.7	206.1	5.7	143.5	5.6
64 s/b (4096)	197.2	5.6	176.8	5.7	139.6	5.7
Uniform BlockGen ($L' \in \{1, 2\}$, uniform weights)						
4 s/b (256)	437.0	5.6	368.5	5.6	222.7	5.5
8 s/b (512)	249.3	5.6	225.1	5.6	129.8	5.4
16 s/b (1024)	156.1	5.5	158.3	5.6	103.5	5.5
32 s/b (2048)	160.0	5.6	136.9	5.6	98.0	5.5
64 s/b (4096)	129.6	5.5	135.3	5.6	101.9	5.6
Uniform BlockGen ($L' \in \{1, 2, 4, 8, 16\}$, almost AR 0.7 - 0.15)						
4 s/b (256)	375.7	5.3	365.7	5.5	234.2	5.5
8 s/b (512)	282.6	5.5	245.9	5.6	160.8	5.5
16 s/b (1024)	180.8	5.4	202.7	5.6	130.9	5.6
32 s/b (2048)	182.8	5.6	173.9	5.6	119.9	5.6
64 s/b (4096)	181.6	5.6	155.2	5.6	113.0	5.6

Table 10: Generative Perplexity (Gen. PPL) and Unigram Entropy with PC sampling for a **Uniform-noise** BlockGen trained on block sizes 1 to 16 (powers of two, uniform weights). **Generation uses block size 16 with AR prefix length 256.** AIPC (d) injects noise into tokens with lowest AR likelihood, improving Gen. PPL over random PC and diffusion-only PC samplers.

(a) Random PC baseline

k_{init}	pc	Gen. PPL ↓	Entropy ↑
NFE/block=4			
0	2	247.4	5.5
1	3	266.0	5.5
NFE/block=8			
0	3	116.9	5.5
6	2	119.8	5.5
NFE/block=16			
0	4	86.4	5.4
12	2	82.8	5.5
NFE/block=32			
0	4	63.9	5.4
20	2	61.3	5.5
26	2	70.4	5.5
NFE/block=64			
0	4	54.1	5.4
40	2	53.7	5.4
52	2	64.3	5.5
58	2	70.3	5.5

(b) Diffusion-only (entropy)

k_{init}	pc	Gen. PPL ↓	Entropy ↑
NFE/block=4			
0	2	308.1	5.5
1	3	266.2	5.5
NFE/block=8			
0	3	124.1	5.4
6	2	133.5	5.5
NFE/block=16			
0	4	77.7	5.4
12	2	96.2	5.5
NFE/block=32			
0	4	49.7	5.3
20	2	63.1	5.5
26	2	75.1	5.5
NFE/block=64			
0	4	35.6	5.2
40	2	54.2	5.4
52	2	71.3	5.5
58	2	78.9	5.6

(c) AR-informed (KLD)

k_{init}	pc	Gen. PPL ↓	Entropy ↑
NFE/block=4			
0	3	415.7	5.7
1	2	364.8	5.6
NFE/block=8			
0	3	143.7	5.5
5	2	126.7	5.5
NFE/block=16			
0	3	90.1	5.5
8	4	82.2	5.5
10	2	79.7	5.5
NFE/block=32			
0	3	53.9	5.4
20	2	63.6	5.5
24	4	70.4	5.5
26	2	70.9	5.5
NFE/block=64			
0	3	45.2	5.4
40	2	55.7	5.5
52	2	64.9	5.5
56	4	69.6	5.5
58	2	72.1	5.5

(d) AR-informed (surprisal)

k_{init}	pc	Gen. PPL ↓	Entropy ↑
NFE/block=4			
0	3	132.0	4.8
1	2	144.0	5.1
NFE/block=8			
0	3	55.5	5.0
5	2	88.6	5.4
NFE/block=16			
0	3	32.3	5.0
8	4	48.2	5.3
10	2	55.8	5.4
NFE/block=32			
0	3	16.1	4.7
20	2	35.8	5.3
24	4	49.4	5.4
26	2	53.1	5.4
NFE/block=64			
0	3	9.3	4.5
40	2	27.3	5.2
52	2	45.3	5.4
56	4	57.5	5.4
58	2	59.0	5.5

Table 11: Generative Perplexity (Gen. PPL) and Unigram Entropy with PC sampling for a **Uniform-noise** BlockGen trained on block sizes 1 to 16 (powers of two, uniform weights). **Generation uses block size 16 with AR prefix length 128.** AIPC (d) injects noise into tokens with lowest AR likelihood, improving Gen. PPL over random PC and diffusion-only PC samplers.

(a) Random PC baseline

k_{init}	pc	Gen. PPL ↓	Entropy ↑
NFE/block=4			
0	2	354.8	5.5
1	3	387.8	5.5
NFE/block=8			
0	3	145.5	5.5
6	2	149.3	5.5
NFE/block=16			
0	4	99.7	5.4
12	2	95.1	5.5
NFE/block=32			
0	4	67.5	5.4
20	2	69.6	5.5
26	2	81.9	5.5
NFE/block=64			
0	4	58.1	5.4
40	2	59.5	5.4
52	2	73.3	5.5
58	2	82.5	5.5

(b) Diffusion-only (entropy)

k_{init}	pc	Gen. PPL ↓	Entropy ↑
NFE/block=4			
0	2	429.7	5.5
1	3	356.9	5.4
NFE/block=8			
0	3	156.7	5.4
6	2	178.0	5.5
NFE/block=16			
0	4	88.2	5.4
12	2	110.3	5.5
NFE/block=32			
0	4	50.1	5.2
20	2	78.6	5.5
26	2	96.7	5.5
NFE/block=64			
0	4	31.2	5.0
40	2	63.6	5.4
52	2	91.0	5.5
58	2	102.2	5.6

(c) AR-informed (KLD)

k_{init}	pc	Gen. PPL ↓	Entropy ↑
NFE/block=4			
0	3	615.9	5.6
1	2	520.1	5.6
NFE/block=8			
0	3	189.0	5.5
5	2	151.9	5.5
NFE/block=16			
0	3	100.7	5.5
8	4	87.7	5.5
10	2	84.4	5.5
NFE/block=32			
0	3	55.2	5.4
20	2	66.3	5.4
24	4	74.4	5.5
26	2	76.2	5.5
NFE/block=64			
0	3	43.3	5.3
40	2	53.9	5.4
52	2	67.4	5.5
56	4	73.1	5.5
58	2	76.8	5.5

(d) AR-informed (surprisal)

k_{init}	pc	Gen. PPL ↓	Entropy ↑
NFE/block=4			
0	3	139.6	4.6
1	2	156.3	4.9
NFE/block=8			
0	3	54.6	4.8
5	2	101.3	5.4
NFE/block=16			
0	3	28.4	4.8
8	4	47.2	5.2
10	2	52.4	5.3
NFE/block=32			
0	3	12.1	4.5
20	2	31.8	5.2
24	4	47.6	5.3
26	2	53.1	5.4
NFE/block=64			
0	3	6.2	4.1
40	2	22.7	5.1
52	2	44.3	5.3
56	4	59.2	5.4
58	2	59.3	5.4

Table 12: Generative Perplexity (Gen. PPL) and Unigram Entropy with PC sampling for a **Uniform-noise** BlockGen trained on block sizes 1 to 16 (powers of two, uniform weights). **Generation uses block size 16 with AR prefix length 64.** AIPC (d) injects noise into tokens with lowest AR likelihood, improving Gen. PPL over random PC and diffusion-only PC samplers.

(a) Random PC baseline

k_{init}	pc	Gen. PPL ↓	Entropy ↑
NFE/block=4			
0	2	437.4	5.5
1	3	471.4	5.5
NFE/block=8			
0	3	164.1	5.4
6	2	169.0	5.5
NFE/block=16			
0	4	106.0	5.4
12	2	105.1	5.5
NFE/block=32			
0	4	69.3	5.3
20	2	73.3	5.4
26	2	85.6	5.5
NFE/block=64			
0	4	57.9	5.4
40	2	60.8	5.4
52	2	77.3	5.5
58	2	89.7	5.5

(b) Diffusion-only (entropy)

k_{init}	pc	Gen. PPL ↓	Entropy ↑
NFE/block=4			
0	2	528.3	5.5
1	3	436.7	5.4
NFE/block=8			
0	3	177.7	5.4
6	2	195.8	5.5
NFE/block=16			
0	4	93.9	5.3
12	2	119.7	5.5
NFE/block=32			
0	4	49.3	5.2
20	2	81.0	5.4
26	2	99.3	5.5
NFE/block=64			
0	4	29.8	5.0
40	2	65.5	5.4
52	2	100.0	5.5
58	2	109.3	5.6

(c) AR-informed (KLD)

k_{init}	pc	Gen. PPL ↓	Entropy ↑
NFE/block=4			
0	3	740.4	5.6
1	2	655.4	5.6
NFE/block=8			
0	3	212.6	5.5
5	2	169.6	5.5
NFE/block=16			
0	3	104.3	5.4
8	4	97.9	5.4
10	2	93.0	5.5
NFE/block=32			
0	3	54.7	5.4
20	2	69.1	5.4
24	4	78.0	5.5
26	2	78.2	5.5
NFE/block=64			
0	3	42.5	5.3
40	2	55.7	5.4
52	2	70.0	5.5
56	4	78.8	5.5
58	2	81.8	5.5

(d) AR-informed (surprisal)

k_{init}	pc	Gen. PPL ↓	Entropy ↑
NFE/block=4			
0	3	133.4	4.3
1	2	149.2	4.6
NFE/block=8			
0	3	53.5	4.7
5	2	106.3	5.3
NFE/block=16			
0	3	25.5	4.7
8	4	46.1	5.2
10	2	53.2	5.3
NFE/block=32			
0	3	10.1	4.4
20	2	30.1	5.1
24	4	46.8	5.3
26	2	52.1	5.3
NFE/block=64			
0	3	4.9	3.9
40	2	21.0	5.0
52	2	43.4	5.3
56	4	62.7	5.4
58	2	61.6	5.4

Table 13: Generative Perplexity (Gen. PPL) and Unigram Entropy with PC sampling for a **Masked** BlockGen trained on block sizes 1 to 16 (powers of two, uniform weights). AIPC (d) remasks tokens with lowest AR likelihood, improving Gen. PPL over random PC and divergence-based selection (c). See Suppl. B.1 for why diffusion-only PC is unsuitable for masked BlockGen.

(a) Random PC baseline

k_{init}	pc	Gen. PPL ↓	Entropy ↑
NFE/block=4			
0	2	473.7	5.8
1	3	515.0	5.8
NFE/block=8			
0	3	311.4	5.8
6	2	273.8	5.7
NFE/block=16			
0	4	236.5	5.7
12	2	184.4	5.7
NFE/block=32			
0	4	209.9	5.7
20	2	161.3	5.7
26	2	138.4	5.7
NFE/block=64			
0	4	180.7	5.7
40	2	139.0	5.7
52	2	121.5	5.7
58	2	114.9	5.7

(b) AR-informed (KLD)

k_{init}	pc	Gen. PPL ↓	Entropy ↑
NFE/block=4			
0	3	547.0	5.8
1	2	678.6	5.7
NFE/block=8			
0	3	381.3	5.7
5	2	249.0	5.7
NFE/block=16			
0	3	201.8	5.5
8	4	204.3	5.6
10	2	142.5	5.6
NFE/block=32			
0	3	100.3	5.3
20	2	81.5	5.5
24	4	101.6	5.6
26	2	104.3	5.6
NFE/block=64			
0	3	48.0	5.1
40	2	54.8	5.4
52	2	82.1	5.6
56	4	99.6	5.6
58	2	97.0	5.6

(c) AR-metric (surprisal)

k_{init}	pc	Gen. PPL ↓	Entropy ↑
NFE/block=4			
0	3	94.9	4.6
1	2	146.5	5.1
NFE/block=8			
0	3	51.9	4.9
5	2	148.2	5.5
NFE/block=16			
0	3	26.5	4.9
8	4	63.4	5.4
10	2	78.1	5.5
NFE/block=32			
0	3	13.6	4.7
20	2	42.3	5.3
24	4	64.5	5.5
26	2	73.0	5.5
NFE/block=64			
0	3	7.5	4.4
40	2	29.6	5.2
52	2	57.3	5.5
56	4	77.3	5.6
58	2	77.1	5.5

Table 14: Generative Perplexity (Gen. PPL) and Unigram Entropy with PC sampling for a **Masked** BlockGen trained on block sizes 1 to 16 (powers of two, uniform weights). **Generation uses block size 16 with AR prefix length 128.** AIPC (d) remasks tokens with lowest AR likelihood, improving Gen. PPL over random PC and divergence-based selection (c). See Suppl. B.1 for why diffusion-only PC is unsuitable for masked BlockGen.

(a) Random PC baseline

k_{init}	pc	Gen. PPL ↓	Entropy ↑
NFE/block=4			
0	2	784.1	5.9
1	3	839.5	5.8
NFE/block=8			
0	3	462.7	5.8
6	2	408.6	5.8
NFE/block=16			
0	4	352.4	5.8
12	2	251.7	5.7
NFE/block=32			
0	4	301.1	5.8
20	2	217.8	5.8
26	2	184.6	5.7
NFE/block=64			
0	4	244.9	5.8
40	2	167.6	5.7
52	2	149.4	5.7
58	2	143.7	5.7

(b) AR-informed (KLD)

k_{init}	pc	Gen. PPL ↓	Entropy ↑
NFE/block=4			
0	3	860.2	5.8
1	2	1026.5	5.7
NFE/block=8			
0	3	551.9	5.7
5	2	374.8	5.7
NFE/block=16			
0	3	283.7	5.5
8	4	272.5	5.6
10	2	184.4	5.6
NFE/block=32			
0	3	109.3	5.2
20	2	90.7	5.5
24	4	123.8	5.6
26	2	118.8	5.6
NFE/block=64			
0	3	44.2	5.0
40	2	57.6	5.4
52	2	91.5	5.6
56	4	119.9	5.6
58	2	114.3	5.6

(c) AR-informed (surprisal)

k_{init}	pc	Gen. PPL ↓	Entropy ↑
NFE/block=4			
0	3	98.6	4.3
1	2	157.5	4.9
NFE/block=8			
0	3	46.7	4.6
5	2	199.0	5.5
NFE/block=16			
0	3	21.7	4.6
8	4	64.2	5.3
10	2	88.3	5.5
NFE/block=32			
0	3	9.1	4.3
20	2	40.5	5.3
24	4	68.1	5.5
26	2	77.2	5.5
NFE/block=64			
0	3	4.8	4.0
40	2	26.5	5.1
52	2	59.7	5.4
56	4	87.4	5.6
58	2	85.2	5.6

Table 15: Generative Perplexity (Gen. PPL) and Unigram Entropy with PC sampling for a **Masked** BlockGen trained on block sizes 1 to 16 (powers of two, uniform weights). **Generation uses block size 16 with AR prefix length 64.** AIPC (d) remasks tokens with lowest AR likelihood, improving Gen. PPL over random PC and divergence-based selection (c). See Suppl. B.1 for why diffusion-only PC is unsuitable for masked BlockGen.

(a) Random PC baseline

k_{init}	pc	Gen. PPL ↓	Entropy ↑
NFE/block=4			
0	2	944.2	5.8
1	3	1043.5	5.9
NFE/block=8			
0	3	559.6	5.8
6	2	517.5	5.8
NFE/block=16			
0	4	421.4	5.8
12	2	301.0	5.8
NFE/block=32			
0	4	366.9	5.8
20	2	250.6	5.8
26	2	203.7	5.7
NFE/block=64			
0	4	297.2	5.8
40	2	212.6	5.8
52	2	174.3	5.7
58	2	159.3	5.7

(b) AR-informed (KLD)

k_{init}	pc	Gen. PPL ↓	Entropy ↑
NFE/block=4			
0	3	1113.9	5.9
1	2	1284.3	5.7
NFE/block=8			
0	3	714.4	5.7
5	2	444.5	5.7
NFE/block=16			
0	3	328.0	5.5
8	4	326.3	5.6
10	2	218.3	5.6
NFE/block=32			
0	3	123.3	5.2
20	2	97.7	5.4
24	4	137.8	5.6
26	2	139.0	5.6
NFE/block=64			
0	3	39.3	4.8
40	2	57.6	5.3
52	2	100.8	5.6
56	4	125.5	5.6
58	2	121.4	5.6

(c) AR-informed (surprisal)

k_{init}	pc	Gen. PPL ↓	Entropy ↑
NFE/block=4			
0	3	79.2	4.0
1	2	152.1	4.7
NFE/block=8			
0	3	39.6	4.3
5	2	227.4	5.5
NFE/block=16			
0	3	19.7	4.5
8	4	65.6	5.3
10	2	89.5	5.4
NFE/block=32			
0	3	7.5	4.2
20	2	39.2	5.2
24	4	69.2	5.4
26	2	82.8	5.5
NFE/block=64			
0	3	3.9	3.8
40	2	23.8	5.0
52	2	61.6	5.4
56	4	88.1	5.5
58	2	88.0	5.6

Table 16: Generative Perplexity (Gen. PPL) and Unigram Entropy with PC sampling for a **Uniform-noise** BlockGen trained on block sizes 1 to 32 (powers of two, uniform weights). **Generation uses block size 32 with AR prefix length 256.** AIPC injects noise into tokens with lowest AR likelihood, improving Gen. PPL over random PC and diffusion-only PC.

(a) Random PC baseline

k_{init}	pc	Gen. PPL ↓	Entropy ↑
NFE/block=4			
0	2	280.4	5.5
1	3	287.3	5.5
NFE/block=8			
0	3	134.7	5.5
6	2	135.4	5.5
NFE/block=16			
0	4	100.1	5.5
12	2	94.0	5.5
NFE/block=32			
0	4	80.3	5.4
20	2	74.8	5.5
26	2	83.6	5.5
NFE/block=64			
0	4	60.2	5.4
40	2	62.0	5.4
52	2	72.9	5.5
58	2	75.8	5.5

(b) Diffusion-only (entropy)

k_{init}	pc	Gen. PPL ↓	Entropy ↑
NFE/block=4			
0	2	336.4	5.5
1	3	275.7	5.4
NFE/block=8			
0	3	151.8	5.4
6	2	147.5	5.5
NFE/block=16			
0	4	88.1	5.4
12	2	106.0	5.5
NFE/block=32			
0	4	55.2	5.2
20	2	76.6	5.5
26	2	87.3	5.5
NFE/block=64			
0	4	30.8	5.0
40	2	61.4	5.4
52	2	79.8	5.5
58	2	87.8	5.5

(c) AR-informed (KLD)

k_{init}	pc	Gen. PPL ↓	Entropy ↑
NFE/block=4			
0	3	431.0	5.6
1	2	450.5	5.6
NFE/block=8			
0	3	178.6	5.5
5	2	150.2	5.5
NFE/block=16			
0	3	91.5	5.4
8	4	92.7	5.5
10	2	90.7	5.5
NFE/block=32			
0	3	67.3	5.4
20	2	71.3	5.5
24	4	77.0	5.5
26	2	78.8	5.5
NFE/block=64			
0	3	47.3	5.3
40	2	59.5	5.5
52	2	71.0	5.5
56	4	74.0	5.5
58	2	74.8	5.5

(d) AR-informed (surprisal)

k_{init}	pc	Gen. PPL ↓	Entropy ↑
NFE/block=4			
0	3	108.3	4.6
1	2	152.4	5.0
NFE/block=8			
0	3	63.0	4.9
5	2	97.7	5.4
NFE/block=16			
0	3	32.4	4.9
8	4	54.7	5.3
10	2	58.5	5.4
NFE/block=32			
0	3	17.7	4.6
20	2	39.1	5.3
24	4	59.4	5.4
26	2	63.2	5.4
NFE/block=64			
0	3	9.0	4.2
40	2	27.9	5.1
52	2	48.4	5.4
56	4	63.1	5.5
58	2	67.1	5.5

Table 17: Generative Perplexity (Gen. PPL) and Unigram Entropy with PC sampling for a **Uniform-noise** BlockGen trained on block sizes 1 to 32 (powers of two, uniform weights). **Generation uses block size 32 with AR prefix length 128.** AIPC injects noise into tokens with lowest AR likelihood, improving Gen. PPL over random PC and diffusion-only PC.

(a) Random PC baseline

k_{init}	pc	Gen. PPL ↓	Entropy ↑
NFE/block=4			
0	2	394.9	5.5
1	3	399.3	5.5
NFE/block=8			
0	3	167.8	5.5
6	2	163.5	5.5
NFE/block=16			
0	4	114.7	5.4
12	2	104.5	5.5
NFE/block=32			
0	4	81.3	5.4
20	2	81.7	5.5
26	2	90.9	5.5
NFE/block=64			
0	4	61.1	5.3
40	2	64.1	5.4
52	2	78.0	5.5
58	2	86.7	5.5

(b) Diffusion-only (entropy)

k_{init}	pc	Gen. PPL ↓	Entropy ↑
NFE/block=4			
0	2	487.1	5.4
1	3	374.5	5.3
NFE/block=8			
0	3	184.1	5.4
6	2	184.5	5.5
NFE/block=16			
0	4	99.4	5.3
12	2	122.7	5.5
NFE/block=32			
0	4	53.2	5.1
20	2	80.0	5.4
26	2	99.8	5.5
NFE/block=64			
0	4	27.2	4.8
40	2	62.1	5.4
52	2	85.9	5.5
58	2	95.9	5.5

(c) AR-informed (KLD)

k_{init}	pc	Gen. PPL ↓	Entropy ↑
NFE/block=4			
0	3	612.9	5.6
1	2	652.5	5.6
NFE/block=8			
0	3	226.8	5.5
5	2	181.4	5.5
NFE/block=16			
0	3	105.5	5.4
8	4	103.1	5.5
10	2	100.4	5.5
NFE/block=32			
0	3	69.6	5.4
20	2	76.0	5.5
24	4	82.6	5.5
26	2	83.8	5.5
NFE/block=64			
0	3	46.5	5.3
40	2	60.4	5.4
52	2	73.5	5.5
56	4	82.2	5.5
58	2	83.9	5.5

(d) AR-informed (surprisal)

k_{init}	pc	Gen. PPL ↓	Entropy ↑
NFE/block=4			
0	3	104.4	4.2
1	2	167.5	4.8
NFE/block=8			
0	3	55.4	4.6
5	2	107.4	5.3
NFE/block=16			
0	3	25.3	4.6
8	4	52.4	5.2
10	2	56.7	5.3
NFE/block=32			
0	3	12.5	4.3
20	2	35.9	5.2
24	4	57.3	5.4
26	2	63.4	5.4
NFE/block=64			
0	3	5.4	3.8
40	2	23.7	5.0
52	2	44.1	5.3
56	4	61.3	5.4
58	2	67.1	5.5

Table 18: Generative Perplexity (Gen. PPL) and Unigram Entropy with PC sampling for a **Uniform-noise** BlockGen trained on block sizes 1 to 64 (powers of two, uniform weights). **Generation uses block size 64 with AR prefix length 256.** AIPC injects noise into tokens with lowest AR likelihood, improving Gen. PPL over random PC and diffusion-only PC.

(a) Random PC baseline

k_{init}	pc	Gen. PPL ↓	Entropy ↑
NFE/block=4			
0	2	266.6	5.5
1	3	270.3	5.5
NFE/block=8			
0	3	130.1	5.5
6	2	129.9	5.5
NFE/block=16			
0	4	94.3	5.5
12	2	88.2	5.5
NFE/block=32			
0	4	76.3	5.4
20	2	76.5	5.5
26	2	78.1	5.5
NFE/block=64			
0	4	63.2	5.4
40	2	64.0	5.5
52	2	71.4	5.5
58	2	77.2	5.5

(b) Diffusion-only (entropy)

k_{init}	pc	Gen. PPL ↓	Entropy ↑
NFE/block=4			
0	2	329.8	5.5
1	3	260.3	5.3
NFE/block=8			
0	3	147.5	5.4
6	2	138.3	5.5
NFE/block=16			
0	4	86.3	5.4
12	2	98.6	5.5
NFE/block=32			
0	4	54.7	5.2
20	2	72.4	5.5
26	2	84.9	5.5
NFE/block=64			
0	4	32.8	5.0
40	2	54.5	5.4
52	2	74.9	5.5
58	2	79.5	5.5

(c) AR-informed (KLD)

k_{init}	pc	Gen. PPL ↓	Entropy ↑
NFE/block=4			
0	3	393.8	5.5
1	2	383.6	5.5
NFE/block=8			
0	3	158.8	5.5
5	2	130.9	5.5
NFE/block=16			
0	3	93.1	5.5
8	4	88.3	5.5
10	2	87.0	5.5
NFE/block=32			
0	3	65.3	5.4
20	2	70.5	5.5
24	4	77.4	5.5
26	2	78.9	5.5
NFE/block=64			
0	3	50.1	5.3
40	2	59.7	5.5
52	2	70.3	5.5
56	4	73.8	5.5
58	2	74.4	5.5

(d) AR-informed (surprisal)

k_{init}	pc	Gen. PPL ↓	Entropy ↑
NFE/block=4			
0	3	123.9	4.6
1	2	161.0	5.0
NFE/block=8			
0	3	57.7	4.7
5	2	99.2	5.4
NFE/block=16			
0	3	33.2	4.8
8	4	53.3	5.3
10	2	57.0	5.3
NFE/block=32			
0	3	18.5	4.6
20	2	39.8	5.3
24	4	57.6	5.4
26	2	63.8	5.5
NFE/block=64			
0	3	9.7	4.1
40	2	29.9	5.2
52	2	49.6	5.4
56	4	63.9	5.5
58	2	66.4	5.5

Table 19: Generative Perplexity (Gen. PPL) and Unigram Entropy with PC sampling for a **Uniform-noise** BlockGen trained on block sizes 1 to 64 (powers of two, uniform weights). **Generation uses block size 64 with AR prefix length 128.** AIPC injects noise into tokens with lowest AR likelihood, improving Gen. PPL over random PC and diffusion-only PC.

(a) Random PC baseline

k_{init}	pc	Gen. PPL ↓	Entropy ↑
NFE/block=4			
0	2	370.1	5.4
1	3	370.0	5.4
NFE/block=8			
0	3	156.5	5.4
6	2	157.4	5.5
NFE/block=16			
0	4	109.3	5.4
12	2	101.3	5.5
NFE/block=32			
0	4	82.7	5.4
20	2	77.1	5.4
26	2	86.2	5.5
NFE/block=64			
0	4	64.6	5.3
40	2	64.0	5.4
52	2	76.1	5.5
58	2	81.8	5.5

(b) Diffusion-only (entropy)

k_{init}	pc	Gen. PPL ↓	Entropy ↑
NFE/block=4			
0	2	453.3	5.4
1	3	351.6	5.3
NFE/block=8			
0	3	177.3	5.3
6	2	171.7	5.5
NFE/block=16			
0	4	99.4	5.3
12	2	111.7	5.5
NFE/block=32			
0	4	53.9	5.1
20	2	80.2	5.4
26	2	90.8	5.5
NFE/block=64			
0	4	28.9	4.8
40	2	56.1	5.4
52	2	78.0	5.5
58	2	86.3	5.5

(c) AR-informed (KLD)

k_{init}	pc	Gen. PPL ↓	Entropy ↑
NFE/block=4			
0	3	568.0	5.5
1	2	554.2	5.5
NFE/block=8			
0	3	198.6	5.4
5	2	164.0	5.5
NFE/block=16			
0	3	102.3	5.4
8	4	99.7	5.5
10	2	97.8	5.5
NFE/block=32			
0	3	66.9	5.4
20	2	72.0	5.5
24	4	82.6	5.5
26	2	84.2	5.5
NFE/block=64			
0	3	47.8	5.3
40	2	60.9	5.4
52	2	72.4	5.5
56	4	78.3	5.5
58	2	80.2	5.5

(d) AR-informed (surprisal)

k_{init}	pc	Gen. PPL ↓	Entropy ↑
NFE/block=4			
0	3	113.6	4.2
1	2	177.2	4.8
NFE/block=8			
0	3	51.4	4.4
5	2	108.0	5.3
NFE/block=16			
0	3	28.2	4.5
8	4	51.5	5.2
10	2	56.7	5.3
NFE/block=32			
0	3	13.1	4.2
20	2	37.3	5.2
24	4	57.3	5.4
26	2	64.8	5.4
NFE/block=64			
0	3	6.5	3.7
40	2	25.2	5.1
52	2	50.6	5.4
56	4	64.1	5.5
58	2	69.8	5.5