BIRTH, DEATH AND DIFF: INSERTION-BASED SEQUENCE GENERATION WITH CORRECTIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Insertion-based masked diffusion models enable flexible-length, parallel generation by interleaving mask insertion and token unmasking. However, because their generation decisions are irreversible, aggressive parallel decoding can introduce errors that later steps cannot fix, forcing conservative decoding with more forward passes (NFEs). We show that, due to multiple alignments, insertion-based models have higher entropy than MDMs and, without a correction mechanism, require even more conservative decoding. To address this, we introduce δMDM, which makes intermediate generations of an insertion-based model revisable by equipping it with substitution and deletion operations. During training, we expose the model to errors produced by aggressive unmasking and insertions to teach it to correct its own mistakes. At inference time, we interleave insertion and unmasking with correction steps that can modify previously unmasked tokens and delete unnecessary masks. This correction mechanism makes aggressive decoding practical, improving both generation speed and quality. δMDM Pareto-dominates FlexMDM, a vanilla insertion-based model, on the speed–quality trade-off. On high-diversity unconditional text generation on OpenWebText, it halves the generative perplexity relative to FlexMDM at high NFEs. It also achieves lower generative perplexity at all inference budgets compared to MDM trained from scratch with variable-length padded sequences. On low-diversity conditional text generation on TinyGSM, it matches FlexMDM accuracy with half as many NFEs.

1 INTRODUCTION

Non-autoregressive language models such as Masked Diffusion Models (MDMs) (Sahoo et al., 2024; Shi et al., 2024; Nie et al., 2025) provide meaningful speedups over autoregressive models (ARMs). However, due to their use of absolute positions, the masked positions far away from the context have high positional uncertainty, i.e., the model may know *what* to say but not *where* to place it in terms of absolute final position (Patel et al., 2025b; Kim et al., 2026). This forces MDMs to generate in a roughly left-to-right manner thereby limiting their parallelism (Nie et al., 2025). Insertion-based Masked Diffusion Models such as FlexMDM (Kim et al., 2025a) grow the sequence incrementally by inserting new `[MASK]` tokens and unmasking any existing `[MASK]` tokens simultaneously. This allows the model to draft by placing important tokens at their approximate positions because insertions allow filling intermediate positions later. However, insertion-based models are also proned to dependency errors under parallel decoding, and they have no mechansim to update positions that have already been unmasked, i.e., they have the irreversible generation problem. Furthermore, when trained with simple denoising objectives, insertion-based models average over possible alignments during training. This converts positional uncertainty into token uncertainty, i.e., each insertion location may correspond to several possible target tokens, producing a diffuse predictive distribution over the vocabulary. For instance, for the sentence *"Your lazy dog jumped over the the black box."* an intermediate state *"Your* `[MASK]` *black."* the `[MASK]` distributes probabilities over all eligible words from the target like *lazy*, *dog*, *jumped*, etc., thereby inflating the predictive entropy. We call this the multiple alignments problem. As a result, the generation needs to be conservative to avoid errors, thereby losing the speed-up advantage of parallel decoding. One way to combat this is to use a non-uniform distribution over alignments during training or equivalently learn the unmasking order

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

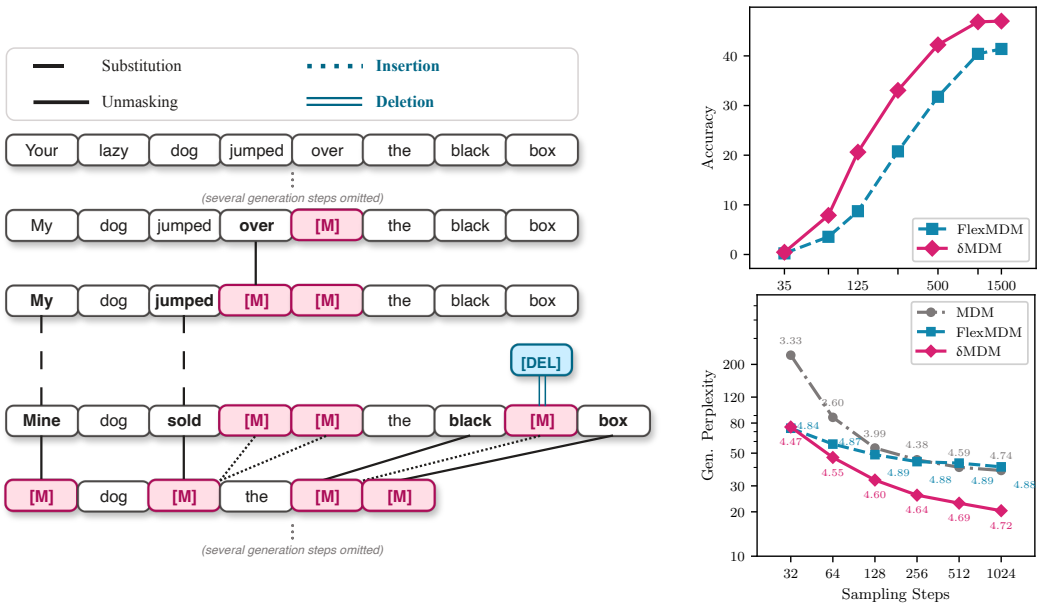


Figure 1: Overview of δMDM generation process. The plots on the right show GSM8K code-execution accuracy and OpenWebText generative perplexity against the sampling budget.

of the target positions using variational learning which tightens the ELBO (Patel et al., 2026b; Zhang et al., 2026). However, this requires an addtional model and could be slow to train as the models get larger.

We take an alternate approach and introduce δMDM (*Delta MDM*), a time-blind insertion-based masked diffusion model that learns to revise its own generations. The name δMDM emphasizes that the model can change both the length (increase or decrease) and content of a partially generated sequence. Its generation process combines insertion (*birth*), deletion (*death*), and token substitution (*diff*). Deletion removes excess mask tokens, while substitutions are performed using token-to-token transitions. The supervision for these operations is derived by forcing the model to predict aggresively to form a state with potential mistakes and then feeding it back to the model to correct the wrong predictions (Figure 1). At inference time, the prediction (*insertion* and *unmasking*), and correction (*deletion* and *substitution*) steps are interleaved. The knowledge that corrections can fix the mistakes in subsequent steps, and time-blindness of the model allows for aggressive drafting in the prediction step, thereby improving both the speed and the quality of generation.

We perform a comprehensive evaluation comparing δMDM with FlexMDM, which does not have a correction mechanism. On the synthetic planning task with controlled complexity, δMDM achieves 10x improvement over FlexMDM on the hardest setting (Appendix C.1). On high-diversity unconditional text generation on OpenWebText, with matched training and inference compute δMDM dominates FlexMDM in terms of both speed and quality without sacrificing diversity. Moreover, δMDM continues to improve with higher inference budgets cutting the generative perplexity by half relative to FlexMDM as well as MDM. At the other extreme, on low-diversity code-based reasoning on TinyGSM, δMDM matches FlexMDM's accuracy with half as many NFEs.

We summarize the key contributions of this work below:

- We introduce a novel self-correction framework for insertion-based masked diffusion models, which is simple and fast to train and does not require any additional auxiliary model.
- We conduct comprehensive evaluation of δMDM across three different type of tasks: controlled complexity synthetic planning, high-diversity unconditional text generation, and low-diverstiy reasoning using code generation, and find that δMDM pareto dominates FlexMDM on the speed–quality trade-off, while also demonstrating superior inference time scaling characteristics for unconditional generation.
- We present ablations demonstrating the importance of our design choices and provide qualitative insights into the behavior of δMDM during generation.

2 PRELIMINARIES

In this section, we introduce the notation and provide a brief overview of insertion-based masked diffusion models (Kim et al., 2025a).

2.1 INSERTION-BASED MASKED DIFFUSION MODELS

Let $\mathbb{X} := \bigcup_{k=1}^{L} \mathbb{V}^k$ be the space of variable length sequences, where $\mathbb{V}$ is the vocabulary including a special mask token denoted by `[M]`. Denote by $p_1 = p_{\text{data}}$ the target data distribution and by p_0 the initial distribution, which in our case is a dirac delta on the empty sequence. The generative process x_t with t going from 0 to 1 is a continuous time Markov chain modeled by its transition rates, which are assumed to be independent per sequence position for tractability. For $t \in [0, 1]$, an intermediate partial sequence x_t is obtained by first sampling a ground truth sequence $x_1 \sim p_1$ and then randomly dropping some positions and masking some according to a schedule. To explain the details, we define some additional notation.

Notation. Define an auxiliary space of partial sequences as $\mathbb{Z} := \bigcup_{k \in [\![L]\!]} \mathbb{V}^k \times \mathbb{S}_k$, where $\mathbb{S}_k := \{(s^1, \ldots, s^k) \in [\![L]\!]^k \mid s^1 < \cdots < s^k\}$ is the space of ordered tuples of positions, L is the maximum length, $[\![L]\!] = \{1, 2, \ldots, L\}$, and superscripts are used to denote index in the sequence. An element of $\mathbb{Z}$ is a pair (x_t, s_t) consisting of a sequence and its corresponding positions in the clean target sequence x_1. Additionally, for $i \in [\![\ell(x_t)]\!]$, let $g_t^i = s_t^i - s_t^{i-1} - 1$ denote the gap length at the i-th gap, with $g_t^1 = s_t^1$ by convention, where $\ell(x_t)$ is the length of x_t. [1] In order to make the exposition cleaner, we will use `[D]` to indicate a dropped position. Note that `[D]` is not part of the vocabulary $\mathbb{V}$ and is never used by the model. With this added notation, we note that the auxiliary space $\mathbb{Z}$ is isomorphic to another space $\bar{\mathbb{Z}} := (\mathbb{V} \cup \{\texttt{[D]}\})^L$, where `[D]` is a special token used to indicate a dropped position. Specifically, the isomorphism is given by the mapping $f_{\text{contract}} : \bar{\mathbb{Z}} \to \mathbb{Z}$ that removes the `[D]` tokens from $\bar{z} \in \bar{\mathbb{Z}}$ to form x, while preserving the exact position information in s to produce $z = (x, s)$.

2.2 TRAINING

For a fixed linear interpolant schedule, an intermediate x_t is obtained as follows. Sample $x_1 \sim p_1$, and $t \sim \text{Uniform}(0, 1)$. Then for each $i \in [\![\ell(x_1)]\!]$, the insertion and unmasking times are sampled as $T_{\text{in}}^i \sim \text{Uniform}(0, 1)$, $T_{\text{um}}^i \sim \text{Uniform}(T_{\text{in}}^i, 1)$. Obtain $\bar{z}_t$ by setting:

$$\bar{z}_t = \begin{cases} \texttt{[D]} & T_{\text{in}} > t \\ \texttt{[M]} & T_{\text{in}} \le t < T_{\text{um}} \\ x_1 & T_{\text{um}} \le t \end{cases} \quad \text{for } i \in \ell(x_1), \qquad \text{and } \bar{z}_t^i = \texttt{[D]} \text{ for } \ell(x_1) < i \le L. \tag{1}$$

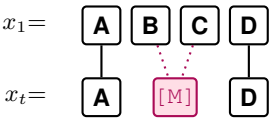
The input to the model is x_t obtained by removing the `[D]` token markers from $\bar{z}_t$, i.e., x_t is the first element of the pair $z_t = (x_t, s_t) = f_{\text{contract}}(\bar{z}_t)$ as defined in the paragraph above. Denote this distribution as $p_t(x_t|x_1)$. The training objective is obtained by minimizing the Bregman divergence between the predicted and target rates. It contains two terms, the insertion loss $\mathcal{L}_{\text{in}}$ and the unmasking loss $\mathcal{L}_{\text{um}}$:

$$\mathcal{L}_{\text{um}}(\theta) = \mathbb{E}\left[\tfrac{1}{1-t} \sum_{i : x_t^i = \texttt{[M]}} -\log K^{\theta,i}(x_1^{s_t^i} | x_t)\right], \quad \mathcal{L}_{\text{in}}(\theta) = \mathbb{E}\left[\tfrac{1}{1-t} \sum_{i=0}^{\ell(x_t)} D\big(g^i(s_t), g^{\theta,i}(x_t)\big)\right]. \tag{2}$$

The expectation in both losses is over $x_1 \sim p_{\text{data}}, t \sim \text{Uniform}(0, 1)$ and $x_t \sim p_t(x_t|x_1)$. Here $D(a, b) = a \log \frac{a}{b} - (a - b)$ is the scalar Bregman divergence. $K^{\theta,i}$ is the predicted token distribution at i-th position, and $g^i(s_t)$ is the number of tokens to be inserted in the i-th gap of x_t according to alignment s_t w.r.t x_1. Therefore, the unmasking loss is nothing but a weighted cross-entropy and the insertion loss simplifies to a regression loss matching the predicted gap length $g^{\theta,i}(x_t)$ to the true $g^i(s_t)$. Note that both the predicted unmasking distribution and the insertion mean condition on the value of t.

[1] In implementation we have a special `[E]` token to indicate the end of the sequence and to compute the gap length of rightmost gap.

Key Limitation: When $\ell(x_t) < \ell(x_1)$, there is at least one [M] in x_t for which the supervision is ambiguous, i.e., that [M] can align with multiple positions in x_1 and therefore the expectation in the loss will result in dispersing the probability mass across multiple vocabulary items. We call this the multiple alignments problem (see Appendix D.1 for its effect on the entropy of the predicted token distributions).



2.3 Generation

The generation proceeds through a tau-leaping (Gillespie, 2001) like procedure that approximately simulates the learned generative CTMC by superposing multiple independent generation decisions at each step. Specifically, given a time grid $0 = t_0 < t_1 < \cdots < t_N = 1$, the generation process proceeds by superposing the following independent decisions at each time step t_n:

1. **Sample Unmaskings**: For each masked position $i \in [\![\ell(x_t)]\!]$ sample the arrival count for each vocabulary item $A_{i,w} \sim \text{Poisson}(\frac{1}{1-t_n} K^{\theta,i}(w|x_t) \cdot \tau_n)$ and sample $x_t^i \sim K^{\theta,i}(\cdot|x_t)$ if $A_{i,w} > 1$ for exactly one $w \in \mathbb{V}$ and leave it as [M] otherwise. We write $\text{SampleUnmask}(x, K, t, \tau)$ for this operation applied to a sequence x with token distributions K at time t and step size τ.
2. **Sample Insertions**: For each gap $j \in [\![\ell(x_t) + 1]\!]$ sample $N_{\text{in}}^j \sim \text{Poisson}(\frac{1}{1-t_n} g^{\theta,j}(x_t) \cdot \tau_n)$ and insert N_{in}^j masks at gap position j in x_t, where $\tau_n = t_n - t_{n-1}$ is the time step size. Insertion is not performed at the last time step t_N, because there will be no unmasking steps after it.

Key Limitation: The generation is iterative but does not have any kind of *undo* mechanism: once a token is unmasked, it cannot be changed even if it is incorrect, and similarly, once a [M] is inserted, it cannot be removed even if it is deemed unnecessary later on. We call this the irreversible generation problem.

3 δMDM

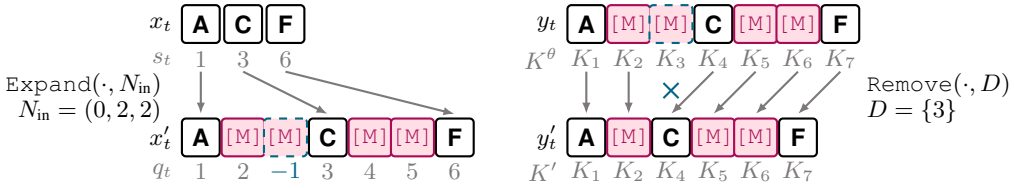
Incorporating corrections can overcome both the limitations mentioned in Section 2. For this, we break down each generation step into two sub-steps: a *proposal* step, which is largely similar to the generation step in Section 2.3, and a newly introduced *correction* step, which is used to undo actions from any of the earlier generation steps that are deemed incorrect or unnecessary. In this section, we describe the training and sampling procedures for δMDM that use proposal and correction steps.[2]

3.1 The Expand and Remove Operators

After insertions, all the positions on the right of the insertion point move further to the right by the number of inserted masks, and this movement compounds with more insertion points. Therefore, in practice, we need to *record* insertions and apply them in one vectorized operation. We describe the vectorized operation in detail here, which greatly simplifies the description of the training and sampling procedures later. Given current $z_t = (x_t, s_t)$, and insertion counts $N_{\text{in}} = (N_{\text{in}}^1, \ldots, N_{\text{in}}^{\ell(x)})$, where N_{in}^i masks are inserted in the i-th gap, (i.e., immediately left of x_t^i), let $\pi(i) \coloneqq i + \sum_{k \le i} N_{\text{in}}^k$ be the position of the current i-th cell after the insertions. Define $\texttt{Expand}(x_t, N_{\text{in}})$ to be the sequence of length $\ell(x_t) + \sum_k N_{\text{in}}^k$ with $\texttt{Expand}(x_t, N_{\text{in}})^{\pi(i)} = x_t^i$ and [M] in all remaining cells. Note that all N_{in}^i are defined in the coordinates of x_t, before any insertion. Conversely, for a set of positions $D \subseteq [\![\ell(x_t)]\!]$, $\texttt{Remove}(y, D)$ deletes the cells in D and moves every cell on their right leftwards. With a slight abuse of notation, we reuse the same operator name when applied to s_t, where it additionally assigns a value to every new cell instead of leaving it empty. Recall that $g_t^i = s_t^i - s_t^{i-1} - 1$ is the number of ground-truth tokens missing in the i-th gap of x_t. As shown in the figure below, the existing entries of s_t move with their tokens through $\texttt{Expand}(s_t, N_{\text{in}})^{\pi(i)} = s_t^i$, and the j-th new cell of the i-th gap ($j = 1, \ldots, N_{\text{in}}^i$, counted from the left) receives $s_t^{i-1} + j$ if $j \le g_t^i$

[2]These names are just intuitive names for the two steps, and are not to be confused with the more formal notions of predictor-corrector diffusion sampler (Song et al., 2021) or the proposal-rejection of Metropolis-Hastings.

and -1 otherwise. Thus new cells are aligned, left to right, to the missing ground-truth positions, and surplus insertions (marked -1) have no ground-truth counterpart. We write $q_t \coloneqq \texttt{Expand}(s_t, N_{\text{in}})$ for this alignment. Likewise, `Remove` applied to per-position model outputs such as K^θ carries each entry along with its token (below, $y'_t = \texttt{Remove}(y_t, D)$ and $K' = \texttt{Remove}(K^\theta, D)$).



3.2 Training

Each training step for δMDM consists of three sub-steps that work sequentially. Figure 2 depicts the training procedure for δMDM.

1 Proposal training. The proposal training step is identical to vanilla insertion-based masked diffusion model as described in Section 2.2.

2 Sampling from the proposal. The goal of this sub-step is to simulate an aggressive proposal, which the correction step can use as the input and learn to correct. For this, we replace the standard τ-leaping sampler step (see Section 2.3) with the following:

1. **Sample Aggressive Unmaskings:** Instead of drawing Poisson arrivals over a step of size τ, which unmask only a random subset of the masks, we unmask *every* masked position in the current x_t at once by greedy decoding, $\tilde{x}_t^i \leftarrow \arg\max_w K^{\theta,i}(w \mid x_t)$ for all i with $x_t^i = \texttt{[M]}$. Addtionally, we also allow already unmasked positions of x_t to also be substituted using the same argmax operation. This is necessary to provide strong training signal for substitutions in the correction step.
2. **Sample Aggressive Insertions:** We sample a scale $\lambda \sim \text{Uniform}(0, 1)$ per sequence and, for every gap i of x_t, an insertion count $N_{\text{in}}^i \sim \text{Poisson}(\lambda\, g^{\theta,i}(x_t))$. Since the insertion rate in Section 2.3 is $\frac{\tau_n}{1-t}\, g^{\theta,i}$, this corresponds to a single leap of size $\tau = \lambda(1-t)$, which ranges from almost no insertions ($\lambda \to 0$) to inserting the entire predicted remaining gap length in one step ($\lambda = 1$). Covering the entire range allows us to adjust the aggressiveness of insertions at inference time.

Therefore, after this step, only the newly inserted masks remain in the proposal sequence.

3 Correction training. Let $y_t \coloneqq \texttt{Expand}(\tilde{x}_t, N_{\text{in}})$ be the proposal from 2. We feed it back into the same network K^θ and train the network to correct it. Note that y_t may be low probablity under $p_t(\cdot|x_1)$ for any time because all masks of x_t have been filled, as if t were close to 1, while the number of new masks depends on the random scale λ. The aggressive proposal therefore destroys the time information, and we have to make K^θ *time-blind*. With that the same network serves as both proposer and corrector. Because $\tilde{x}_t$ is obtained by argmax and N_{in} by sampling, no gradient flows through the construction of y_t, which serves only as an on-policy input. The targets come from the alignment $q_t = \texttt{Expand}(s_t, N_{\text{in}})$ as shown in Fig. 2. Every cell with $q_t^i \neq -1$ has the ground-truth counterpart $x_1^{q_t^i}$, and every cell with $q_t^i = -1$ is a surplus insertion whose target is the deletion token `[DEL]`. In other words, within each gap, the new masks are matched from left to right to the missing ground-truth tokens, and the excess masks at the right end of the gap are deleted. Note that many other alignments are valid, but we fix this one for simplicity. The total correction loss is the sum of a substitution and deletion terms:

$$\mathcal{L}_{\text{subs}}(\theta) = \mathbb{E}\left[\frac{1}{1-t}\sum_{i:q_t^i \neq -1} -\log K^{\theta,i}(x_1^{q_t^i} | y_t)\right], \quad \mathcal{L}_{\text{del}}(\theta) = \mathbb{E}\left[\frac{1}{1-t}\sum_{i:q_t^i = -1} -\log K^{\theta,i}(\texttt{[DEL]} | y_t)\right]. \quad (3)$$

Unlike $\mathcal{L}_{\text{um}}$, which only covers masked positions, $\mathcal{L}_{\text{subs}}$ covers every cell of y_t that has a ground-truth counterpart. The model therefore learns to replace wrong argmax fills, keep correct tokens (including the ones already unmasked in x_t), and unmask the new masks that align to ground truth. $\mathcal{L}_{\text{del}}$ teaches it to delete masks that have no ground-truth counterpart. The full training objective is $\mathcal{L} = \mathcal{L}_{\text{um}} + \mathcal{L}_{\text{in}} + \mathcal{L}_{\text{subs}} + \mathcal{L}_{\text{del}}$, where each term is normalized per-token across the whole batch.

Algorithm: δMDM Training Step

Require: Dataset p_{data}, parameters θ, learning rate η

1: ❶ Sample $x_1 \sim p_{\text{data}}$, $t \sim \text{Unif}(0,1)$
2: $\forall i \in [\![\ell(x_1)]\!]$ sample
3: $T^i_{\text{in}} \sim \text{Unif}(0,1)$ $T^i_{\text{um}} \sim \text{Unif}(T^i_{\text{in}}, 1)$
4: Get $\bar{z}_t$ using Eq. (1) and $(x_t, s_t) \leftarrow f_{\text{contract}}(\bar{z}_t)$
5: $\mathcal{L} \leftarrow \mathcal{L}_{\text{um}}(x_1, x_t, s_t; \theta) + \mathcal{L}_{\text{in}}(x_1, x_t, s_t; \theta)$
6: ❷ $\lambda \sim \text{Unif}(0,1)$
7: $\forall i : [\![\ell(x_t)]\!]$, set $\tilde{x}^i_t \leftarrow \arg\max_w K^{\theta,i}(w \mid x_t)$
8: $N^i_{\text{in}} \sim \text{Poisson}(\lambda\, g^{\theta,i}(x_t))$ for every gap i of x_t
9: $y_t \leftarrow \texttt{Expand}(\tilde{x}_t, N_{\text{in}})$
10: ❸ $q_t \leftarrow \texttt{Expand}(s_t, N_{\text{in}})$
11: $\mathcal{L} \leftarrow \mathcal{L}_{\text{del}}(x_1, y_t, q_t; \theta) + \mathcal{L}_{\text{subs}}(x_1, y_t, q_t; \theta)$
12: Update $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}(\theta)$

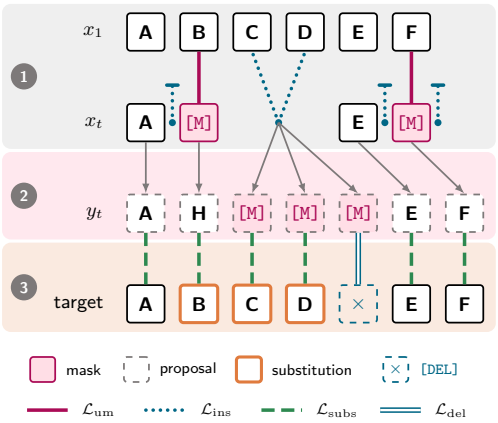


Figure 2: **δMDM Training Step.** *Left:* training algorithm; *right:* toy example. The three shaded blocks in the algorithm correspond to stages 1–3 of the illustration. ❶ ***Proposal training*** as described in Section 2.2. ❷ ***Sample proposal*** by filling all positions with argmax of K^θ and inserting from a Poisson count with mean between 0 and $g^{\theta,i}(x_t)$ for each gap. All counts are drawn in the coordinates of x_t, and inserting them shifts every later token to the right (`Expand`, Section 3.1). ❸ ***Correction training*** by aligning y_t to x_1 via $q_t = \texttt{Expand}(s_t, N_{\text{in}})$ (Section 3.1): each new cell is aligned to a missing ground-truth position, or gets -1 and is deleted by predicting `[DEL]`. Then we substitute tokens ($\mathcal{L}_{\text{subs}}$), fill the leftmost new `[M]`s, and delete the excess `[M]`s from the right ($\mathcal{L}_{\text{del}}$).

3.3 Generation

For generation, we interleave proposal and correction steps in an alternating fashion. The sampler starts from the empty sequence $x_0 = \emptyset$ and applies the proposal-correction step depiced in Algorithm 2 over a time grid $0 = t_0 < \cdots < t_n = 1$, returning x_n. Using predicted gap lengths $g^\theta(x_t)$ we insert the number of masks at the corresponding gap positions (line 3) to obtain x'_t. We allow for increasing insertion intensity by using $\lambda > 1$ as the insertion scale factor. As in training, the proposal y_t for correction is formed by taking an argmax over the vocabulary items at every masked in x'_t. Then the corrector (Algorithm 2) performs another forward pass to produce $K^\theta(.|y_t), g^\theta(y_t)$, using the output to substitute incorrect tokens, delete unnecessary newly inserted masks, and produces updated unmasking distributions at all masked positions. For substitutions (line 3), we use argmax over the vocabulary items from $K^\theta(.|y_t)$ at all the unmasked positions $\mathcal{U} = \{i \mid x'^i_t \neq \texttt{[M]}\}$. For deletions (line 1), we consider only the newly inserted masks $\mathcal{M}_{\text{new}}$, and if a deletion event is sampled for any of these positions, we remove the position and shift every later cell to the left along with the corresponding logits (line 4-5). The output of the correction is the new sequence $x_{t+\tau}$ and the improved unmasking distribution $K^\theta(.|x_{t+\tau})$ on all the masked positions. The unmasking is then performed based on these updated logits with a chosen sampling function. On steps where correction is skipped, $x_{t+\tau} = x'_t$ and the logits are carried through the same shift, $\texttt{Expand}(K^\theta(\cdot \mid x_t), N_{\text{in}})$ as shown in line 9 of Algorithm 1. In this case, the newly inserted cells have no logits yet and are therefore not unmasked in that step. Both proposal and correction steps are counted towards the total number of steps. Note that since the model is time-blind, the only place where the value of t is used is in the Poisson sampler for the insertions and deletions.

4 Experiments

Setup All models use a transformer backbone with rotary positional embeddings, pre-layernorm and are trained from scratch. [3] For masked diffusion we use the time-agnostic version with unweighted cross-entropy loss (Zheng et al., 2024) as it has been shown to work better than time-conditioned MDLM that uses ELBO (Sahoo et al., 2026). Moreover, we pre-train MDM for variable length sequences using padded sequences instead of packed sequences for a fair comparison. For vanilla insertion-based masked diffusion we reimplement FlexMDM (Kim et al., 2025a), which is not

[3] We use the transformer implementation provided in Patel et al. (2025a). See Appendix B for more details.

Algorithm 1: δMDM Sampling Step.

Require: $t, \tau, x_t, \theta, \lambda$, *apply_correction*

$\tilde{x}_t \leftarrow x_t$

$\forall i : [\![\ell(x_t)]\!]$, set $\tilde{x}_t^i \leftarrow \arg\max_w K^{\theta,i}(w \mid x_t)$

$\forall i \in [\![\ell(x_t)]\!]$, $N_{\text{in}}^i \sim \text{Poisson}(\frac{\lambda\tau}{1-t}\, g^{\theta,i}(x_t))$

$x'_t \leftarrow \texttt{Expand}(x_t, N_{\text{in}}), \quad y_t \leftarrow \texttt{Expand}(\tilde{x}_t, N_{\text{in}})$

if *apply_correction* **then**

  $\mathcal{U} \leftarrow \{i \mid x_t'^i \neq \texttt{[M]}\}, \quad \mathcal{M}_{\text{new}} \leftarrow \{i \mid y_t^i = \texttt{[M]}\}$

  $x_{t+\tau}, \hat{K} \leftarrow \text{Correction}(x'_t, y_t, t, \tau, \mathcal{U}, \mathcal{M}_{\text{new}})$

else

  $x_{t+\tau} \leftarrow x'_t, \quad \hat{K} \leftarrow \texttt{Expand}(K^\theta(\cdot \mid x_t), N_{\text{in}})$

end if

$x_{t+\tau} \leftarrow \text{SampleUnmask}(x_{t+\tau}, \hat{K}, t, \tau)$

return $x_{t+\tau}$

Algorithm 2: Correction.

Require: x'_t, y_t, time t, step size τ, unmasked positions $\mathcal{U}$, new mask positions $\mathcal{M}_{\text{new}}$

$\forall i \in \mathcal{M}_{\text{new}}$, $N_{\text{del}}^i \sim \text{Poisson}(\frac{\tau}{1-t}\, K^{\theta,i}(\texttt{[DEL]} | y_t))$, $\mathcal{M}_{\text{del}} \leftarrow \{i \in \mathcal{M}_{\text{new}} \mid N_{\text{del}}^i \geq 1\}$

Set $\hat{x} \leftarrow x'_t$

$\forall i \in \mathcal{U}$, set $\hat{x}^i \leftarrow \arg\max_w K^{\theta,i}(w | y_t)$

$x_{t+\tau} \leftarrow \texttt{Remove}(\hat{x}, \mathcal{M}_{\text{del}})$

$\hat{K} \leftarrow \texttt{Remove}(K^\theta(\cdot | y_t), \mathcal{M}_{\text{del}})$

return $x_{t+\tau}, \hat{K}$

Model	Gen. PPL (↓)						Entropy (↑)					
	32	64	128	256	512	1024	32	64	128	256	512	1024
MDM	230.96	87.48	54.18	45.08	40.10	38.16	3.33	3.59	3.98	4.32	4.56	4.71
ILM	-	-	222.19	53.70	51.21	49.74	-	-	4.27	4.42	4.30	4.42
FlexMDM	**73.80**	57.59	48.83	43.81	42.80	40.37	**4.83**	**4.87**	**4.89**	**4.88**	**4.88**	**4.88**
δMDM	75.23	**46.76**	**32.85**	**25.94**	**22.89**	**20.35**	4.47	4.55	4.60	4.64	4.69	4.72

Table 1: Generative perplexity and token entropy under GPT-2 Large at different sampling budgets.

time-blind and uses adaptive layer normalization conditioned on a time embedding. In order to obtain time-blind δMDM, we pin the time embedding to a constant value of 1, but keep the same architecture as FlexMDM. [4] Since δMDM uses two forward passes per training step, we compare against baselines that are trained for twice the number of total training steps at the same global batch size. We conduct comprehensive evaluation on three different type of tasks: controlled complexity synthetic planning, high-diversity unconditional text generation, and low-diverstiy reasoning using code generation. The rest of this section discusses the results on the text generation task with results on synthetic star-graph planning task provided in Appendix C.1.

4.1 Unconditional Generation on OpenWebText

We pretrain MDM, FlexMDM and δMDM on a preprocessed version of OpenWebText (OWT) dataset tokenized using the GPT-2 tokenizer, where we remove documents that are longer than 1024 tokens, and pad the remaining documents to 1024 tokens creating a variable length corpus (Patel et al., 2026a). δMDM is train for 400k steps with global batch size of 512, while MDM, ILM and FlexMDM are trained for 800k steps. We generate 1000 samples per setting with nucleus sampling applied to all methods with $p = 0.9$ and report generation perplexity under GPT-2 Large and as well as token entropy across total sampling step budgets.

As shown in Table 1, δMDM achieves the lower generative perplexity compared to FlexMDM at all budgets except for $T = 32$. Interestingly, we observe that δMDM keeps improving in quality at a higher rate compared to FlexMDM as the inference budget increases (see Fig. 1 bottom right). Specifically, at higher NFEs, δMDM obtains 50% lower generative perplexity compared to FlexMDM and MDM. Conversely, δMDM at 128 sampling steps obtains lower Gen. PPL than FlexMDM at 1024 steps, an 8x speedup. The token entropy for δMDM is generally lower than that of FlexMDM but it remains stable across all budgets. [5] Figure 4 shows the effect of the insertion scale λ at inference: for every budget, Gen. PPL is lowest for moderate scales ($\lambda \approx 1.1$–1.25) and rises sharply for $\lambda \geq 1.5$, where the generated sequences approach the maximum length. We analyze the length of the generated sequences in Appendix D.6.

[4] The impact of removing time from the model's input is negligible even for FlexMDM. See Appendix D.2 for more details.

[5] Note that the token entropies ranges are lower than the ones reported in the the literature on MDMs because we train on variable length sequences which results in lower average entropy.

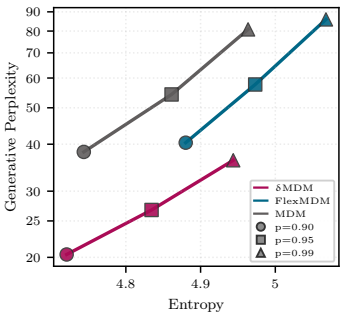


Figure 3: Entropy vs. generation perplexity (GPT-2 Large) on OWT at T=512 for nucleus sampling with $p \in \{0.9, 0.95, 0.99\}$.

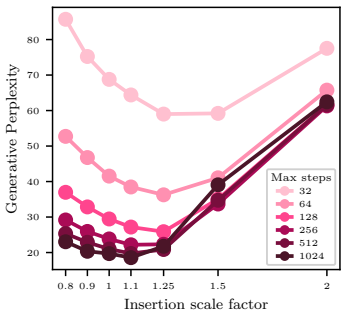


Figure 4: Generation perplexity (GPT-2 Large) of δMDM on OWT vs. insertion scale λ for different sampling budgets T.

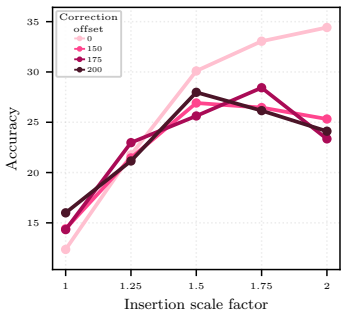


Figure 5: GSM8K pass@1 of δMDM vs. insertion scale λ at T=250 (other budgets in Appendix D.5).

4.2 Conditional Generation on TinyGSM

We also train from scratch on TinyGSM (Liu et al., 2023) dataset containing 11.8M grade school math problems converted to solutions in Python. This dataset is qualitatively different from OWT in several ways: (1) it a conditional generation task, where there is an input question, and (2) since the solutions are Python code, the output sequences are much more structured and show less diversity. We use the same model architectures and training budgets as we do for OWT pretraining. Once trained, we evaluate the models on the test split of GSM8K (Cobbe et al., 2021) dataset containing 1.32K problems and report the pass@1 score by comparing exact numerical answers. For this task, we use greedy decoding, i.e., we pick the token with the highest probability to fill the selected mask position for all the models. The pass@1 accuraries are reported in Table 2. δMDM performs significantly better than FlexMDM at all budgets with the largest difference at low budgets where δMDM the same performance as FlexMDM at half the steps.

Model	125	250	500	1000	1500
MDM	**37.38**	**40.71**	40.33	40.64	41.09
FlexMDM	8.72	20.77	31.77	40.41	41.39
δMDM	20.62	33.06	**42.23**	**46.85**	**47.01**

Table 2: GSM8K test pass@1 (%) vs. sampling budget T. We compare MDM and FlexMDM with our method at 400k training steps.

4.3 Analysis

In this section, we provide a more detailed analysis of the corrections performed by δMDM.

When do corrections happen and how frequently? We measure correction intensity as the fraction of eligible positions that are substituted or deleted at each step, and plot it against normalized generation time (Figs. 10 and 11, Appendix D.4). On both OWT and GSM8K, substitutions peak early and deletions dominate near the end. Early in generation the sequence is short and many tokens were predicted with little context, so replacing them in place is the natural fix. Later, most of the sequence is already fixed, and errors tend to show up as redundant or misplaced tokens, which are easier to remove than to rewrite.

What happens when we increase the aggressiveness of insertions? We can increase the aggressiveness of insertions by increasing the insertion scale λ at inference time. As shown in Fig. 4, for unconditional generation, there is a diminishing return in quality as we increase the insertion scale especially at higher inference budgets. At small budgets ($T \leq 64$), Gen. PPL keeps improving up to λ=1.25 and stays flat at λ=1.5, whereas for $T \geq 256$ it is lowest at $\lambda \approx 1.1$–1.25 and rises sharply beyond. On GSM8K, we additionally vary the correction offset, i.e., the number of prediction steps taken before correction steps begin (Fig. 5, and Appendix D.5 for $T \in \{500, 1000\}$). When correction is applied from the start (offset 0), accuracy increases with λ up to λ=2 at T=250 and up to λ=1.75 at $T \in \{500, 1000\}$. In contrast, with a late correction start, accuracy peaks at λ=1.5

and then declines, so the gap to offset 0 widens as insertions become more aggressive, reaching roughly 10 points or more at the largest λ for every budget. At λ=1 this ordering is reversed for $T \in \{250, 500\}$, where a late start is slightly better, i.e., early corrections pay off mainly when the proposal is aggressive enough to introduce errors. Larger budgets raise accuracy across all settings, but do not remove the need for early correction: aggressive insertion pays off most when the model has enough correction steps to fix the resulting mistakes.

What is the impact of correction step on predictive entropy? The correction step performs a second forward pass over the sequence given the aggressive proposal we plot the average predictive entropy for each mask slot during the proposal and correction step (Fig. 8 in Appendix D.1) and find that the entropy is generally lower after the correction step. This provides an intutive understanding of how proposal-correction mechanism helps accelerate generation.

5 Related Work

There have been serveral approaches proposed for incorporating corrections in fixed-length MDMs. Wang et al. (2026) propose re-masking decoded tokens based on decoding confidence. More recent works such as (Kim et al., 2025b; Huang et al., 2025; Liu et al., 2026) add additional heads over pretrained MDMs to estimate token quality in a partially masked sequence, where lower-quality tokens are re-masked for the next unmasking step. Supervision for the correction heads is obtained by randomly replacing correct tokens with other tokens in the vocabulary (Huang et al., 2025), sampling from the model's own token distribution (Kim et al., 2025b), or introducing human-injected synthetic errors (Ji et al., 2025). Pretrained MDMs are then fine-tuned to optimize the correction loss with MDM loss regularization. Schiff et al. (2026) propose alternating unmasking and correction steps similar to our approach. However, they limit the corrections to substitutions only, and don't consider length changing operations. Von Rütte et al. (2025) uses a mixed noise schedule with both uniform vocabulary and masked noise, thereby enabling the model to perform token flips. (Zhao et al., 2025) also correct decoded tokens but require architectural changes to the denoising model, thereby necessitating retraining. While much of the prior work on correction focuses on fixed-length MDMs, few approaches address self-correction in variable-length settings. (Wu et al., 2026) introduces length control in MDMs using custom expansion and deletion operations that adaptively insert and remove masks. However, they focus on code infilling benchmarks and do not consider substitutions.

6 Limitations

The corrector is trained on a single step of aggressive proposal. At inference, proposals are built on the model's own earlier outputs, so errors can compound over many proposal-correction rounds and produce states that were not seen in training resulting in exposure bias. Unrolling several rounds during training or fully on-policy training would could address this, but it would considerably more expensive. The corrections can be made more expressive by allowing deletion of visible tokens, however, that would cause ambiguous supervision in the current framework.

Further scaling and adaptation study is left as future work.

7 Conclusion

This work introduces δMDM, a variable-length masked diffusion model with insertion, deletion, and substitution actions. It learns the correction operations from its own aggressive proposals without requiring any auxiliary model. δMDM Pareto-dominates FlexMDM on the speed–quality trade-off, exhibits better scaling with inference budget across unconditional as well as conditional text generation tasks.

AI use statement

In this work, we used generative AI tools for some parts of the model implementation and analysis code. Additionally, we used generative AI tools for formatting some of the figures and tables, and for polishing the writing of some parts of the manuscript. The two main authors have reviewed all AI

generated code as well as content, and all the authors take full responsibility for the final content of this work, including text, claims and artifacts produced with the aid of generative AI.

Reproducibility statement

We provide complete details of our core algorithm in Section 3. Section 4 describes the datasets used, along with the details of preprocessing. The model architecture is also described in Section 4, with key hyperparameters listed in Appendix B. We implement all the baselines described in Section 4 using the xLM framework for non-autoregressive language modeling (Patel et al., 2025a). Anonymized source code is available at `https://anonymous.4open.science/r/DeltaMDM`. The trained model checkpoints and logged training loss curves on Weights and Biases will be made publicly available upon acceptance.

References

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Daniel T Gillespie. Approximate accelerated stochastic simulation of chemically reacting systems. *The Journal of chemical physics*, 115(4):1716–1733, 2001.

Zemin Huang, Yuhang Wang, Zhiyang Chen, and Guo-Jun Qi. Don't settle too early: Self-reflective remasking for diffusion language models, 2025. URL https://arxiv.org/abs/2509.23653.

Yatai Ji, Teng Wang, Yuying Ge, Zhiheng Liu, Sidi Yang, Ying Shan, and Ping Luo. From denoising to refining: A corrective framework for vision-language diffusion model, 2025. URL https://arxiv.org/abs/2510.19871.

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, September 2025a. URL http://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.

Seunggeun Kim, Jaeyeon Kim, Taekyun Lee, Yuyuan Chen, Yilun Du, Sham Kakade, and Sitan Chen. From interface to inference: Eliciting any-order inference from any-order models, 2026. URL https://arxiv.org/abs/2607.26504.

Bingbin Liu, Sebastien Bubeck, Ronen Eldan, Janardhan Kulkarni, Yuanzhi Li, Anh Nguyen, Rachel Ward, and Yi Zhang. Tinygsm: achieving ¿80 URL https://arxiv.org/abs/2312.09241.

Liming Liu, Binxuan Huang, Zixuan Zhang, Xin Liu, Bing Yin, and Tuo Zhao. Backplay: Head-only look-back self-correction for diffusion language models, 2026. URL https://arxiv.org/abs/2601.06428.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan LI. Large language diffusion models. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 50608–50646. Curran Associates, Inc., 2025. doi: 10.52202/085713-1689. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/48b383b24230e0e6e649d9c98dae4d8c-Paper-Conference.pdf.

Dhruvesh Patel, Durga Prasad Maram, Sai Sreenivas Chintha, Benjamin Rozonoyer, and Andrew McCallum. XLM: A Python package for non-autoregressive language models, December 2025a. URL http://arxiv.org/abs/2512.17065.

Dhruvesh Patel, Tahira Naseem, Gaurav Pandey, Md Arafat Sultan, Andrew McCallum, and Ramón Fernandez Astudillo. Improved sampling from masked diffusion models with position contrastive guidance. In *NeurIPS 2025 Workshop on Structured Probabilistic Inference & Generative Modeling*, 2025b. URL https://openreview.net/forum?id=e0WmOrWbtc.

Dhruvesh Patel, Aishwarya Sahoo, Avinash Amballa, Tahira Naseem, Tim G. J. Rudner, and Andrew McCallum. Insertion language models: Sequence generation with arbitrary-position insertions, 2025c. URL https://arxiv.org/abs/2505.05755.

Dhruvesh Patel, Benjamin Rozonoyer, Soumitra Das, Tahira Naseem, Tim G. J. Rudner, and Andrew McCallum. A continuous time markov chain framework for insertion language models. In *The 29th International Conference on Artificial Intelligence and Statistics*, 2026a. URL https://openreview.net/forum?id=nCyV21FmUI.

Dhruvesh Patel, Benjamin Rozonoyer, Gaurav Pandey, Tahira Naseem, Ramón Fernandez Astudillo, and Andrew McCallum. Insertion based sequence generation with learnable order dynamics. *ArXiv*, abs/2602.18695, 2026b. URL https://api.semanticscholar.org/CorpusID:285973138.

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

Subham Sekhar Sahoo, Jean-Marie Lemercier, Zhihan Yang, Justin Deschenaux, Jingyu Liu, John Thickstun, and Ante Jukic. Scaling beyond masked diffusion language models, 2026. URL https://arxiv.org/abs/2602.15014.

Yair Schiff, Omer Belhasin, Roy Uziel, Guanghan Wang, Marianne Arriola, Gilad Turok, Ran Zilberstein, Michael Elad, and Volodymyr Kuleshov. Learn from Your Mistakes: Self-Correcting Masked Diffusion Models, February 2026. URL http://arxiv.org/abs/2602.11590.

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.

Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-Based Generative Modeling through Stochastic Differential Equations, February 2021. URL http://arxiv.org/abs/2011.13456.

Dimitri Von Rütte, Janis Fluri, Yuhui Ding, Antonio Orvieto, Bernhard Schölkopf, and Thomas Hofmann. Generalized interpolating discrete diffusion. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 61810–61843. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/von-rutte25a.html.

Guanghan Wang, Yair Schiff, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Remasking discrete diffusion models with inference-time scaling, 2026. URL https://arxiv.org/abs/2503.00307.

Zirui Wu, Lin Zheng, Zhihui Xie, Jiacheng Ye, Jiahui Gao, Shansan Gong, Yansong Feng, Zhenguo Li, Wei Bi, Guorui Zhou, and Lingpeng Kong. Dreamon: Diffusion language models for code infilling beyond fixed-size canvas, 2026. URL https://arxiv.org/abs/2602.01326.

Yangtian Zhang, Zhe Wang, Arthur Gretton, Rex Ying, David van Dijk, Michalis K. Titsias, and Jiaxin Shi. Variational learning for insertion-based generation, 2026. URL https://arxiv.org/abs/2606.02133.

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.

Kaiwen Zheng, Yongxin Chen, Hanzi Mao, Ming-Yu Liu, Jun Zhu, and Qinsheng Zhang. Masked Diffusion Models are Secretly Time-Agnostic Masked Models and Exploit Inaccurate Categorical Sampling. In *The Thirteenth International Conference on Learning Representations*, October 2024. URL https://openreview.net/forum?id=CTC7CmirNr.

APPENDIX

A Summary of Notation **13**

B Experimental Details **15**

C Additional Tasks **15**

C.1 Synthetic Planning: Star Graphs 15

D Additional Analysis **15**

D.1 Multiple alignment problem 15

D.2 Time-blindness 16

D.3 Entropy vs. generation perplexity across sampling budgets 17

D.4 How are corrections distributed? 17

D.5 GSM8K accuracy vs. insertion scale 19

D.6 Length of generated sequences on OWT 19

E Extended Related Work **19**

A SUMMARY OF NOTATION

Table 3 summarizes the notation used throughout the paper.

Notation	Description
	Sequences and spaces
$[\![n]\!]$	The set $\{1, 2, \dots, n\}$
$\mathbb{V}$	Vocabulary of tokens, including the mask token `[M]`
`[M]`	Mask token
`[D]`	Marker for a dropped position; not part of $\mathbb{V}$ and never seen by the model
`[DEL]`	Deletion token; the correction target for surplus inserted masks
L	Maximum sequence length
$\mathbb{X}$	Space of variable length sequences, $\mathbb{X} = \bigcup_{k=1}^{L} \mathbb{V}^k$
$\mathbb{Z}$	Space of partial sequences with positions, $\mathbb{Z} = \bigcup_{k \in [\![L]\!]} \mathbb{V}^k \times \mathbb{S}_k$, where $\mathbb{S}_k := \{(s^1, \dots, s^k) \in [\![L]\!]^k \mid s^1 < \cdots < s^k\}$
$\bar{\mathbb{Z}}$, f_{contract}	$\bar{\mathbb{Z}} = (\mathbb{V} \cup \{\texttt{[D]}\})^L$, and the isomorphism $f_{\text{contract}} : \bar{\mathbb{Z}} \to \mathbb{Z}$ that removes the `[D]` markers from $\bar{z}$ and records the positions of the remaining tokens
$\ell(x)$	Length of the sequence x
x^i	i-th token of the sequence x (superscripts index positions)
	Noising and training
$p_1 = p_{\text{data}}$, p_0	Data distribution; initial distribution (a Dirac delta on the empty sequence)
t	Time, $t \in [0, 1]$
x_1, x_t	Clean sequence; noisy partial sequence at time t
$\bar{z}_t$, $z_t = (x_t, s_t)$	Noisy state in $\bar{\mathbb{Z}}$ (Eq. (1)) and its contraction $f_{\text{contract}}(\bar{z}_t)$
s_t	Positions in x_1 of the tokens of x_t
T_{in}^i, $T_{\text{um}}{}^i$	Insertion and unmasking times of the i-th position
g_t^i	Number of ground-truth tokens missing in the i-th gap of x_t, $g_t^i = s_t^i - s_t^{i-1} - 1$
$\mathcal{L}_{\text{um}}$, $\mathcal{L}_{\text{in}}$	Unmasking and insertion losses (Eq. (2))
$\mathcal{L}_{\text{subs}}$, $\mathcal{L}_{\text{del}}$	Substitution and deletion losses (Eq. (3))
$D(a, b)$	Scalar Bregman divergence, $D(a, b) = a \log \frac{a}{b} - (a - b)$
	Model
θ	Model parameters
$K^{\theta,i}(\cdot \mid x)$	Predicted token distribution at the i-th position of x, over $\mathbb{V} \cup \{\texttt{[DEL]}\}$
$g^{\theta,i}(x)$	Predicted length of the i-th gap of x
	Proposal, correction, and sampling
$N_{\text{in}} = (N_{\text{in}}^1, \dots)$	Number of masks inserted in each gap, in the coordinates of x_t
λ	Insertion scale; $\lambda \sim \text{Uniform}(0, 1)$ during training and $\lambda > 1$ is allowed at inference
$\texttt{Expand}(\cdot, N_{\text{in}})$, $\pi(i)$	Inserts masks into the gaps, shifting later cells to the right; $\pi(i)$ is the new index of cell i (Section 3.1)
$\texttt{Remove}(\cdot, D)$	Deletes the cells in D, shifting later cells to the left
$\tilde{x}_t$	x_t with every `[M]` filled by the argmax of K^θ
x'_t	x_t after insertions, $x'_t = \texttt{Expand}(x_t, N_{\text{in}})$
y_t	Proposal given to the corrector, $y_t = \texttt{Expand}(\tilde{x}_t, N_{\text{in}})$
q_t	Alignment of y_t to x_1, $q_t = \texttt{Expand}(s_t, N_{\text{in}})$; $q_t^i = -1$ marks a surplus mask with target `[DEL]`
$\mathcal{U}$, $\mathcal{M}_{\text{new}}$, $\mathcal{M}_{\text{del}}$	Unmasked positions; newly inserted masks; new masks selected for deletion
N_{del}^i	Deletion arrival count at position i
$0 = t_0 < \cdots < t_N = 1$, τ	Sampling time grid; step size
$\text{SampleUnmask}(x, K, t, \tau)$	τ-leaping unmasking step (Section 2.3)

Table 3: Summary of notation used throughout the paper.

B Experimental Details

All trained models use the architecture and optimization settings in Table 4. δMDM is trained for 400k steps, while MDM, ILM, and FlexMDM are trained for 800k steps. OpenWebText sequences are padded to length 1024, and TinyGSM sequences use length 512.

Hyperparameter	**Value**
Layers	12
Hidden size	768
Attention heads	12
Feed-forward width	3072
Rotary dimension	64
Dropout	0.1
Activation	ReLU
Non-embedding parameters	$\approx$ 87M
Optimizer	AdamW
Learning rate	10^{-4}
Weight decay	0.01
Schedule	Constant with linear warmup
Warmup steps	2,000
Global batch size	512
Gradient clipping	1
Precision	bfloat16 mixed

Table 4: Architecture and optimization settings shared by all trained models. The parameter count is the transformer body and excludes token embeddings.

C Additional Tasks

C.1 Synthetic Planning: Star Graphs

Given a start node and a target node, the model must generate the unique path on a star-shaped graph. Following (Patel et al., 2025c), we use three splits (easy, medium, hard) with increasing difficulty in terms of variable arm lengths and non-junction start nodes. The model input is a flattened edge list followed by the start node, target node, and EOS token. We use an insertion scale factor of 2 (the best value from a sweep over I) and report exact match (EM) at sampling budget $T{=}500$ (Table 5). MDM uses confidence based decoding with the top probability value for choosing unmasking positions in the sequence during sampling. FlexMDM uses the same inference procedure detailed in (Kim et al., 2025a).

Model	$\text{Star}_{\text{easy}}$ (↑)	$\text{Star}_{\text{medium}}$ (↑)	$\text{Star}_{\text{hard}}$ (↑)
MDM	100	100	74.38
FlexMDM	92.00	93.13	7.27
Ours	98.70	98.60	96.47

Table 5: Star-graph exact match (%) at $T{=}500$ sampling steps.

D Additional Analysis

D.1 Multiple alignment problem

Figure 7 shows the entropy of the predicted token distribution at masked positions as a function of time t on OWT for FlexMDM and MDM. Due to the multiple alignments problem (Section 2.2), the supervision for a `[M]` in FlexMDM is spread across several candidate positions of x_1, resulting in much higher entropy than MDM for most of the generation. Figure 8 shows the same quantity for δMDM at sampling budget $T{=}512$, before and after the correction step.

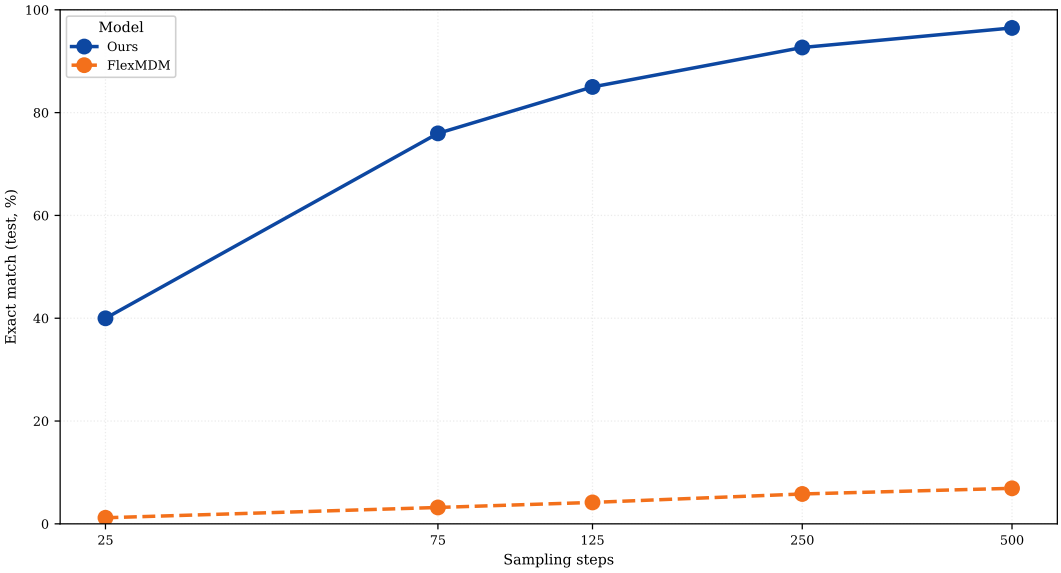


Figure 6: $\text{Star}_{\text{hard}}$ exact match (↑) vs. sampling budget for δMDM and FlexMDM at $T \in \{32, 64, 128, 256, 512\}$.

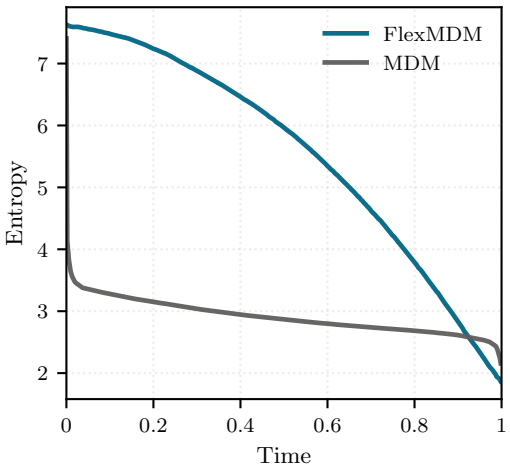


Figure 7: Entropy of the predicted token distribution at masked positions vs. time t on OWT for FlexMDM and MDM.

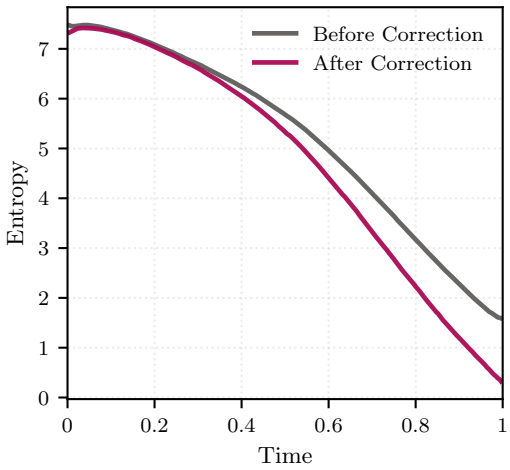


Figure 8: Entropy of the predicted token distribution at masked positions vs. time t on OWT for δMDM at T=512, before and after correction.

D.2 TIME-BLINDNESS

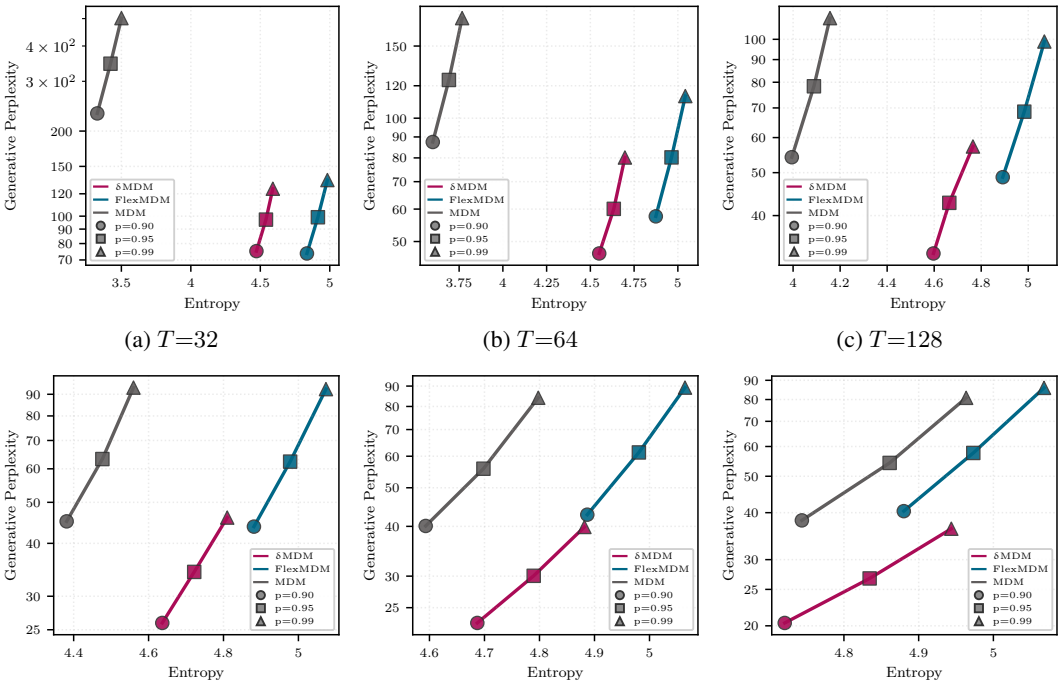
Table 6 compares FlexMDM with time-blind FlexMDM on GSM8K with both models trained for 800k steps. As seen here, removing time from model's input does make a large difference in performance.

Model	250	500	1000
FlexMDM	20.77	31.77	40.41
FlexMDM (time-blind)	21.00	31.84	38.13

Table 6: GSM8K test pass@1 (%) vs. sampling budget T for FlexMDM and time-blind FlexMDM, both trained for 800k steps.

D.3 Entropy vs. generation perplexity across sampling budgets

Figure 3 in the main text reports entropy vs. Gen. PPL at T=512. Here we include the corresponding plots for the remaining budgets $T \in \{32, 64, 128, 256, 1024\}$.



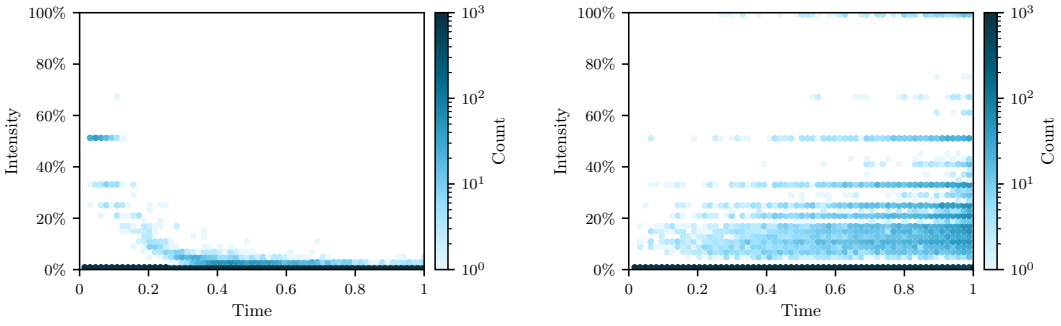
(a) T=32 (b) T=64 (c) T=128

(d) T=256 (e) T=512 (f) T=1024

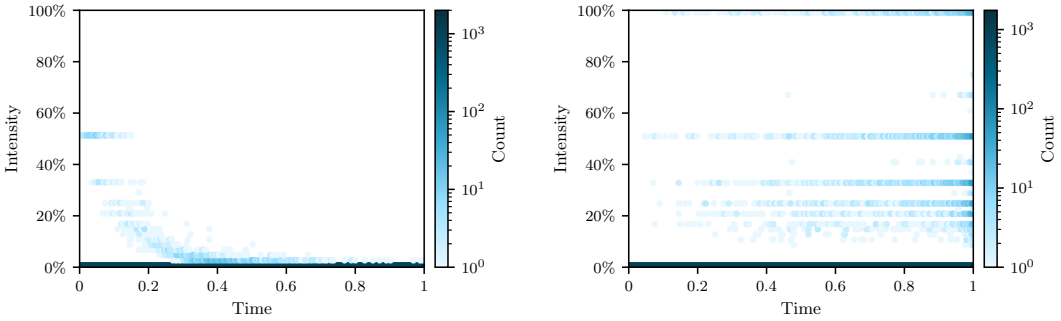
Figure 9: Entropy vs. generation perplexity (GPT-2 Large) on OWT for nucleus sampling with $p \in \{0.9, 0.95, 0.99, 1\}$ at the remaining sampling budgets.

D.4 How are corrections distributed?

Figures 10 and 11 show the distribution of correction intensity over normalized time. The intensity is defined as the percentage of eligible positions that are corrected (substituted or deleted) at that time step. Each cell counts how often a given intensity occurs at that time across all the generations. As shown in the figures, the substitutions peak early in the generation process while the deletions increase towards the end. This intuitively makes sense because isolated substitutions are difficult to make towards the end of the generation when most of the sequence is already generated. Moreover, early on there isn't much need for deletions because the sequence is still short.

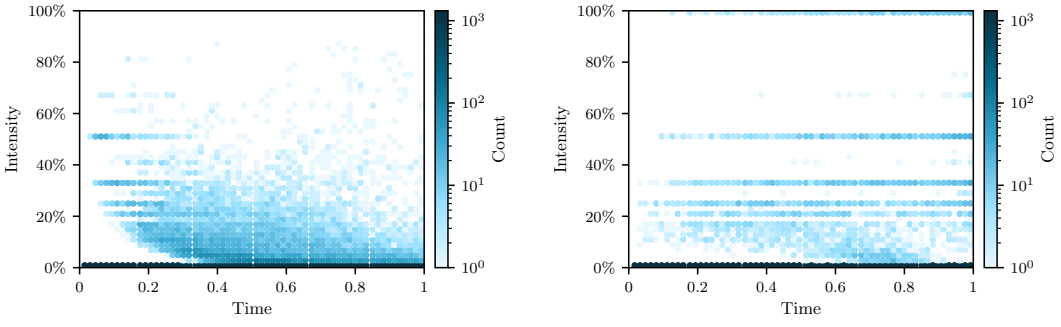


(a) Substitutions, T=128 (b) Deletions, T=128

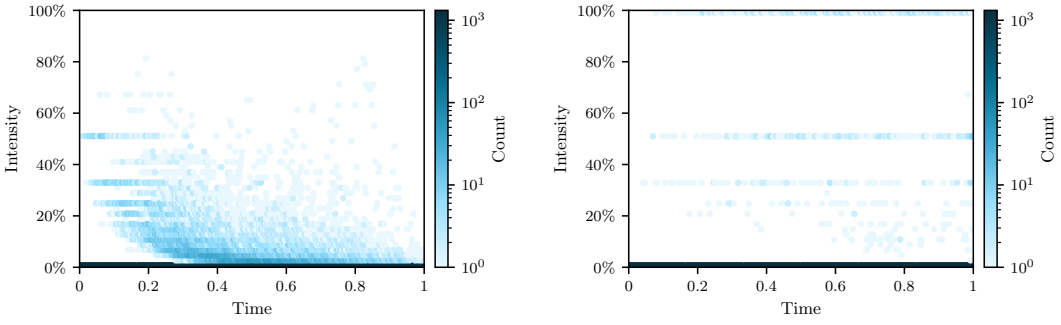


(c) Substitutions, T=512 (d) Deletions, T=512

Figure 10: Distribution of substitution and deletion intensity over normalized time for δMDM on OWT.



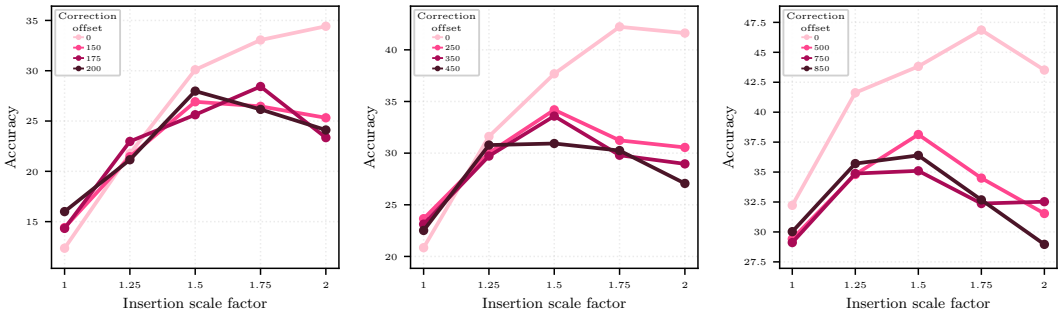
(a) Substitutions, T=125 (b) Deletions, T=125



(c) Substitutions, T=500 (d) Deletions, T=500

Figure 11: Distribution of substitution and deletion intensity over normalized time for δMDM on GSM8K.

D.5 GSM8K ACCURACY VS. INSERTION SCALE



(a) T=250 (b) T=500 (c) T=1000

Figure 12: GSM8K pass@1 vs. insertion scale at sampling budgets $T \in \{250, 500, 1000\}$.

D.6 LENGTH OF GENERATED SEQUENCES ON OWT

Figure 13 compares the quantiles of the generated sequence lengths against the OWT validation set, and Fig. 14 shows how the mean length of δMDM generations changes with the insertion scale λ.

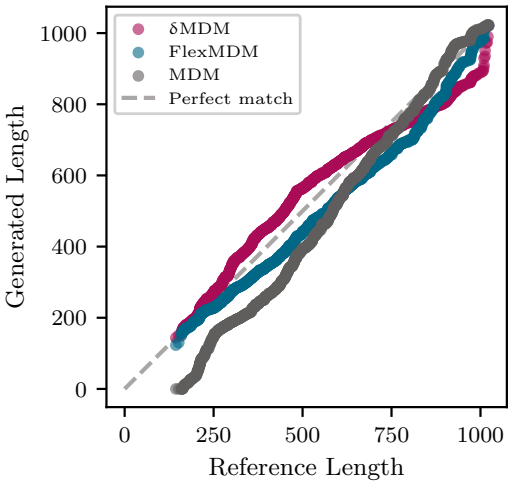


Figure 13: Q–Q plot of the sequence length quantiles of δMDM, FlexMDM, and MDM generations vs. OWT validation.

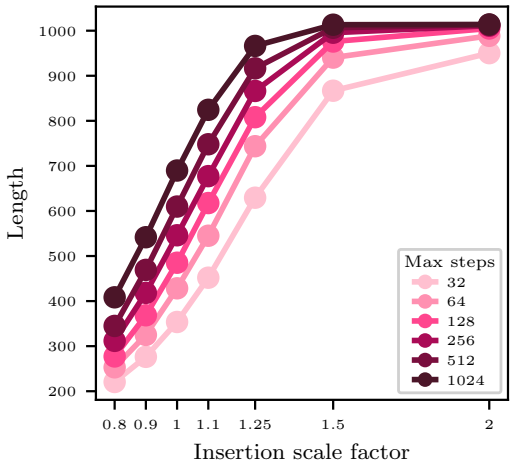


Figure 14: Mean length of δMDM generations on OWT vs. insertion scale λ for different sampling budgets T.

E EXTENDED RELATED WORK

Remasking: Prior correction approaches in MDMs involve re-masking decoded tokens based on decoding confidence, where tokens are re-masked during inference in proportion to the negative of this probability (Wang et al., 2026). More recent works such as (Kim et al., 2025b; Huang et al., 2025; Liu et al., 2026) add additional heads over pretrained MDMs to estimate token quality in a partially masked sequence, where lower-quality tokens are re-masked for the next unmasking step. Supervision for the correction heads is obtained by randomly replacing correct tokens with other tokens in the vocabulary (Huang et al., 2025), sampling from the model's own token distribution (Kim et al., 2025b), or introducing human-injected synthetic errors (Ji et al., 2025). Pretrained MDMs are then fine-tuned to optimize the correction loss with MDM loss regularization.

Non-remasking Correction: Contrary to re-masking approaches, (Schiff et al., 2026) proposes alternating unmasking and correction steps. The corrector is conditioned on a more informative

sequence obtained by unmasking all positions, and then modifies incorrectly decoded tokens while readjusting the predictive distributions over masked positions. Pretrained MDMs can be fine-tuned to learn this correction, though it requires an additional forward pass during inference. Von Rütte et al. (2025) uses a mixed noise schedule with both uniform vocabulary and masked noise, thereby enabling the model to perform token flips. (Zhao et al., 2025) also correct decoded tokens but require architectural changes to the denoising model, thereby necessitating retraining. (SeedDiffusion)

Variable Length Correction: While much of the prior work on correction focuses on fixed-length MDMs, few approaches address self-correction in variable-length settings. (Wu et al., 2026) performs length control using custom expansion and deletion operations that adaptively insert and remove masks, achieving superior performance on code infilling benchmarks.