

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

Durga Prasad Maram^{1*} Dhruvesh Patel^{1*} Tahira Naseem² Benjamin Rozonoyer¹
Gaurav Pandey² Ramón Fernandez Astudillo² Andrew McCallum¹

¹University of Massachusetts Amherst ²IBM Research

ABSTRACT

Insertion-based masked diffusion models enable flexible-length, parallel generation by interleaving mask insertion with token unmasking. However, because their generation decisions are irreversible, aggressive parallel decoding can introduce errors that subsequent steps cannot correct, necessitating conservative decoding with more forward passes (NFEs). Moreover, they exhibit higher entropy than standard MDMs because they marginalize over multiple alignments during training and therefore require even more conservative decoding in the absence of a correction mechanism. To address this limitation, we introduce δ MDM, which makes intermediate generations revisable by equipping an insertion-based model with substitution and deletion operations. During training, we expose the model to errors caused by aggressive unmasking and insertion, teaching it to correct its own mistakes. At inference time, we interleave insertion and unmasking with correction steps that can revise previously unmasked tokens and remove unnecessary masks. This mechanism makes aggressive decoding practical, improving both generation speed and quality. δ MDM Pareto-dominates FlexMDM, a vanilla insertion-based MDM, on the speed–quality trade-off. On high-diversity unconditional text generation on OpenWebText, it halves the generative perplexity relative to FlexMDM at the highest inference budget. 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 speedups over autoregressive models (ARMs) by producing more tokens per forward pass. However, because they use absolute positions, the masked positions far from the context have high positional uncertainty, i.e., the model may know *what* to say but not *where* to place it (Patel et al., 2025b; Kim et al., 2026b). This forces MDMs to generate in a block-wise 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, since later insertions can fill intermediate positions. However, insertion-based models are also prone to dependency errors under parallel decoding, and they have no mechanism 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. 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, in the sentence “Your lazy dog jumped over the black box.”, an intermediate state “Your [MASK] black.” forces the [MASK] to distribute probability over all eligible target tokens, such as *lazy*, *dog*, and *jumped*, thereby inflating the predictive entropy. We call this the multiple alignments problem. As

*Equal contribution.

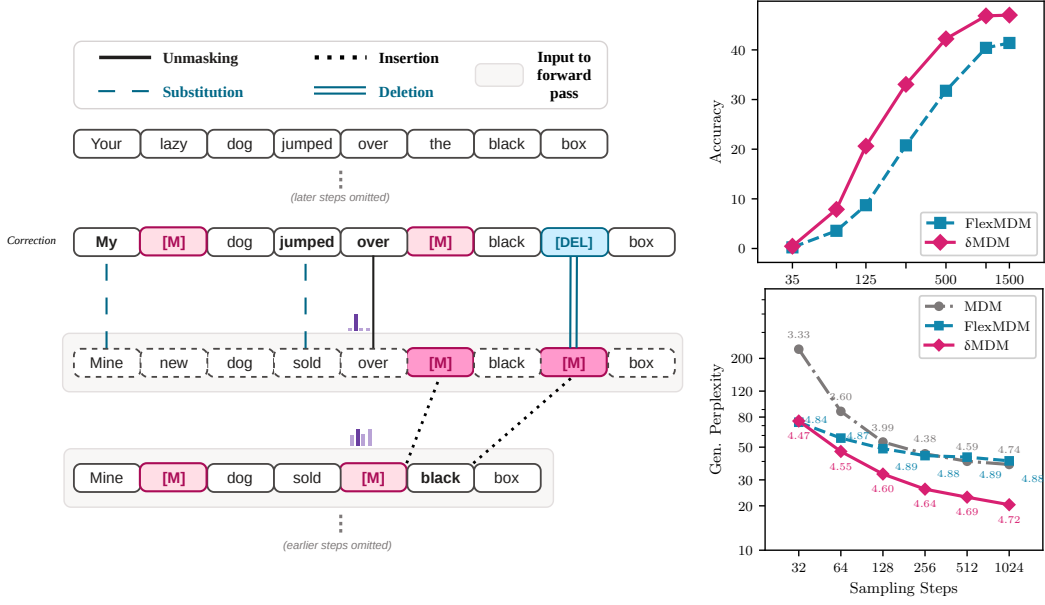


Figure 1: Overview of the δ MDM generation process. The plots on the right show GSM8K code-execution accuracy and OpenWebText generative perplexity versus the sampling budget. Numeric annotations on the OpenWebText plot are token-entropy values.

a result, confidence-based decoding must stay conservative to avoid errors, sacrificing the speedup 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 of the target positions using variational learning, which tightens the ELBO (Patel et al., 2026b; Zhang et al., 2026). However, this requires an additional model and can be slow to train as the models get larger.

We take an alternative approach and introduce δ MDM (*Delta MDM*), a time-blind insertion-based masked diffusion model that learns to revise its own generations. The name δ MDM emphasizes that the model can change both the length (increase or decrease) and the 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 aggressively 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, we interleave the prediction (*insertion* and *unmasking*) and correction (*deletion* and *substitution*) steps. The knowledge that corrections can fix mistakes in subsequent steps and the model’s time-blindness allow aggressive drafting in the prediction step, thereby improving both the speed and the quality of generation.

We perform a comprehensive evaluation comparing δ MDM with FlexMDM, which does not have a correction mechanism. On the synthetic planning task with controlled complexity, δ MDM achieves a $10\times$ improvement over FlexMDM in the hardest setting (Appendix C.1). On high-diversity unconditional text generation on OpenWebText, with matched training and inference compute δ MDM Pareto-dominates FlexMDM in terms of both speed and quality. Moreover, δ MDM keeps improving as the inference budget grows: at 1024 steps it halves the generative perplexity of FlexMDM and reduces that of MDM by 47%, and with only 128 steps it already outperforms both baselines at 1024 steps. At the other extreme, on low-diversity code-based reasoning on TinyGSM, δ MDM 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 requires no auxiliary model.
- We conduct a comprehensive evaluation of δ MDM across three types of tasks: controlled-complexity synthetic planning, high-diversity unconditional text generation, and low-diversity reasoning via code generation, and find that δ MDM Pareto-dominates FlexMDM on the speed–

quality trade-off, while also demonstrating superior inference-time scaling for unconditional generation.

- We present ablations demonstrating the importance of our design choices and provide qualitative insights into the behavior of δ MDM 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 $\mathbb{Z} := \bigcup_{k \in \llbracket L \rrbracket} \mathbb{V}^k \times \mathbb{S}_k$, where $\mathbb{S}_k := \{(s^1, \dots, s^k) \in \llbracket L \rrbracket^k \mid s^1 < \dots < s^k\}$ is the space of ordered tuples of positions, L is the maximum length, $\llbracket L \rrbracket = \{1, 2, \dots, L\}$, 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 . Let $\ell(x_t)$ denote the length of x_t . Every sequence carries a special [E] token at position $\ell(x_t) + 1$, which is never dropped, masked, or deleted, and we set $s_t^0 := 0$ and $s_t^{\ell(x_t)+1} := \ell(x_1) + 1$. For $i \in \llbracket \ell(x_t) + 1 \rrbracket$, the i -th gap is the slot immediately left of x_t^i , and $g_t^i = s_t^i - s_t^{i-1} - 1$ is its length, i.e., the number of tokens of x_1 missing in that slot. Note that the rightmost gap is the one immediately left of [E] . 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 $\bar{\mathbb{Z}} := (\mathbb{V} \cup \{\text{[D]}\})^L$. Specifically, the isomorphism is given by the mapping $f_{\text{contract}} : \bar{\mathbb{Z}} \rightarrow \mathbb{Z}$ 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 $i \in \llbracket \ell(x_1) \rrbracket$, 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:

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

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 $\mathcal{L}_{\text{um}}$:

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

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 , and $g^{\theta,i}(x_t)$ is read off the cell at position i , which is the [E] cell for the rightmost gap. 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 .

Key Limitation: The model observes x_t but not the alignment s_t . In a fixed-length MDM, the i -th mask always corresponds to x_1^i , so the only uncertainty in its target is the content of x_1 . Once positions are dropped, the same x_t can arise from the same x_1 under several alignments. In Fig. 2, $x_t = (A, [M], D)$ arises from $x_1 = (A, B, C, D)$ with either $s_t = (1, 2, 4)$ or $\tilde{s}_t = (1, 3, 4)$. Each sample supervises a single token, but the minimizer of Eq. (2) is the mixture $\sum_s p(s_t=s | x_t) p(x_1^s | x_t, s_t=s)$, which spreads probability mass over every token the mask could align to (here B and C). The gap targets $g^i(s_t)$ are ambiguous in the same way. We call this the multiple alignments problem (see Appendix D.1 for its effect on the entropy of the predicted token distributions).

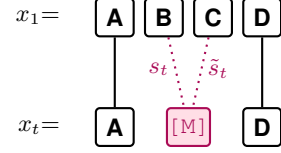


Figure 2: Two alignments of the same x_t to the same x_1 : $s_t = (1, 2, 4)$ and $\tilde{s}_t = (1, 3, 4)$.

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 $0 = t_0 < t_1 < \dots < t_N = 1$, the generation process proceeds by superposing the following independent decisions at each time step t_n :

1. **Sample Unmaskings:** For each masked position $i \in \llbracket \ell(x_t) \rrbracket$, sample the arrival count for each vocabulary item $A_{i,w} \sim \text{Poisson}(\frac{1}{1-t_n} K^{\theta,i}(w|x_t) \cdot \tau_n)$. If $\sum_w A_{i,w} = 1$, set x_t^i to the unique w with $A_{i,w} = 1$; otherwise leave the position as [M]. 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 $j \in \llbracket \ell(x_t) + 1 \rrbracket$ 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 in the j -th gap of x_t (immediately left of x_t^j), 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 δ MDM

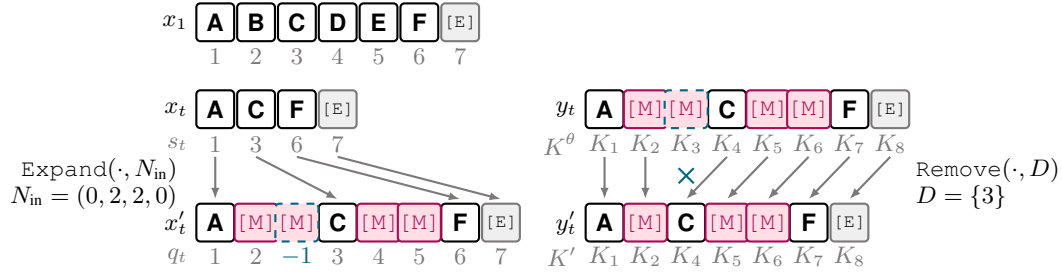
Incorporating corrections can overcome both the limitations mentioned in Section 2. For this, we break down each generation step into two sub-steps: a *proposal* step, which is largely similar to the generation step in Section 2.3, and a newly introduced *correction* step, which is used to undo actions from any of the earlier generation steps that are deemed incorrect or unnecessary. In this section, we describe the training and sampling procedures for δ MDM that use proposal and correction steps.¹

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

¹These 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.

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 $N_{\text{in}} = (N_{\text{in}}^1, \dots, N_{\text{in}}^{\ell(x_t)+1})$, where N_{in}^i masks are inserted in the i -th gap (i.e., immediately left of x_t^i), let $\pi(i) := i + \sum_{k \leq i} N_{\text{in}}^k$ be the position of the current i -th cell after the insertions, for $i \in [\ell(x_t) + 1]$. Define $\text{Expand}(x_t, N_{\text{in}})$ to be the sequence of length $\ell(x_t) + \sum_k N_{\text{in}}^k$ with $\text{Expand}(x_t, N_{\text{in}})^{\pi(i)} = x_t^i$ 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 sequence y and a set of positions $D \subseteq [\ell(y)]$, $\text{Remove}(y, D)$ deletes the cells in D and moves every cell on their right leftwards ($[E]$ is never in D). 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 from Section 2 that g_t^i 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 $\text{Expand}(s_t, N_{\text{in}})^{\pi(i)} = s_t^i$, and the j -th new cell of the i -th gap ($j = 1, \dots, N_{\text{in}}^i$, counted from the left) receives $s_t^{i-1} + j$ if $j \leq g_t^i$ 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. In the figure, $x_1 = \text{ABCDEF}$. x_t keeps A , C , and F , aligned to positions 1, 3, and 6, and the second mask inserted before C is marked -1 because that gap contains only the missing token B . Arrows show the index movement under Expand and the logit movement under Remove . We write $q_t := \text{Expand}(s_t, N_{\text{in}})$ for this alignment. Likewise, Remove applied to per-position model outputs such as K^θ carries each entry along with its token (below, $y_t' = \text{Remove}(y_t, D)$ and $K' = \text{Remove}(K^\theta, D)$).



3.2 TRAINING

Each training step for δMDM consists of three sub-steps that work sequentially. Figure 3 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, $\hat{x}_t^i \leftarrow \arg \max_w K^{\theta, i}(w | x_t)$ for all i with $x_t^i = [M]$. Additionally, 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 an insertion fraction $\rho \sim \text{Uniform}(0, 1)$ per sequence and, for every gap i of x_t , an insertion count $N_{\text{in}}^i \sim \text{Poisson}(\rho g^{\theta, i}(x_t))$. 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 $\tau = \rho(1-t)$, which ranges from almost no insertions ($\rho \rightarrow 0$) to inserting the entire predicted remaining gap length in one step ($\rho = 1$). Thus ρ is the fraction of the predicted remaining gap that is inserted in one step, and covering $(0, 1)$ exposes the corrector to every per-step insertion fraction that the sampler can produce (see Section 3.3).

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

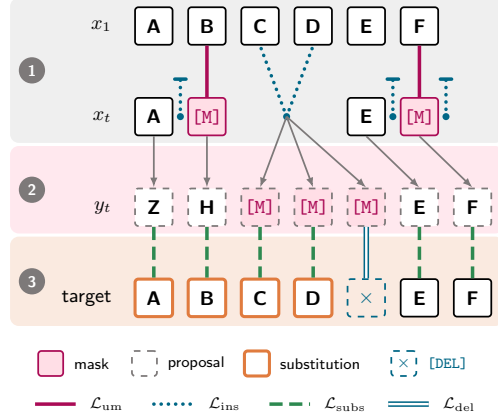
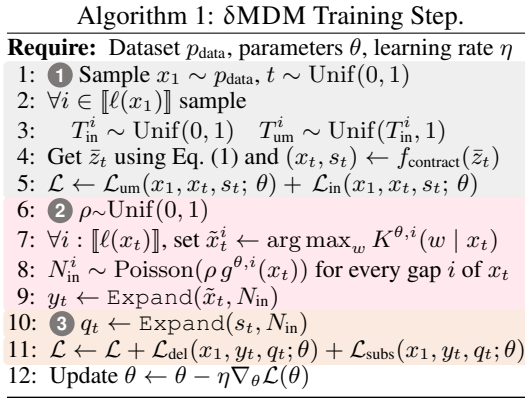


Figure 3: **δ MDM Training Step.** *Left:* Algorithm 1; *right:* toy example. The three shaded blocks in the algorithm correspond to stages 1–3 of the illustration. **1 Proposal training** as described in Section 2.2. **2 Sample proposal** by filling every position with the argmax of K^{θ} , including tokens already unmasked in x_t , 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). In the example the argmax replaces the already-unmasked A with Z , and the correction target at that cell remains A . **3 Correction training** by aligning y_t to x_1 via $q_t = \text{Expand}(s_t, N_{\text{in}})$ (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 Correction training. Let $y_t := \text{Expand}(\tilde{x}_t, N_{\text{in}})$ 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 probability 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 fraction ρ . 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 $q_t = \text{Expand}(s_t, N_{\text{in}})$ as shown in Algorithm 1. 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:

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

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.

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 described in Algorithm 2 over a time grid $0 = t_0 < \dots < t_n = 1$, returning x_n . The first forward pass, on x_t , gives both the greedy fill and the insertion counts (lines 2–3). As in training, the greedy fill $\tilde{x}_t$ takes the argmax over the vocabulary at every position of x_t , and the counts N_{in}^i are sampled from the predicted gap lengths $g^{\theta, i}(x_t)$. Unlike the training fraction ρ , the insertion scale λ multiplies the

Algorithm 2: δ MDM Sampling Step.

Require: $t, \tau, x_t, \theta, \lambda, \text{apply_correction}$

- 1: $\tilde{x}_t \leftarrow x_t$
- 2: $\forall i : \llbracket \ell(x_t) \rrbracket$, set $\tilde{x}_t^i \leftarrow \arg \max_w K^{\theta, i}(w | x_t)$
- 3: $\forall i \in \llbracket \ell(x_t) + 1 \rrbracket$, $N_{\text{in}}^i \sim \text{Poisson}(\frac{\lambda\tau}{1-t} g^{\theta, i}(x_t))$
- 4: $x'_t \leftarrow \text{Expand}(x_t, N_{\text{in}})$, $y_t \leftarrow \text{Expand}(\tilde{x}_t, N_{\text{in}})$
- 5: **if** *apply_correction* **then**
- 6: $\mathcal{U} \leftarrow \{i | x_t'^i \neq [\text{M}]\}$, $\mathcal{M}_{\text{new}} \leftarrow \{i | y_t^i = [\text{M}]\}$
- 7: $x_{t+\tau}, \hat{K} \leftarrow \text{Correction}(x'_t, y_t, t, \tau, \mathcal{U}, \mathcal{M}_{\text{new}})$
- 8: **else**
- 9: $x_{t+\tau} \leftarrow x'_t$, $\hat{K} \leftarrow \text{Expand}(K^{\theta}(\cdot | x_t), N_{\text{in}})$
- 10: **end if**
- 11: $x_{t+\tau} \leftarrow \text{SampleUnmask}(x_{t+\tau}, \hat{K}, t, \tau)$
- 12: **return** $x_{t+\tau}$

Algorithm 3: Correction.

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

- 1: $\forall i \in \mathcal{M}_{\text{new}}$,
 $N_{\text{del}}^i \sim \text{Poisson}(\frac{\tau}{1-t} K^{\theta, i}([\text{DEL}] | y_t))$,
 $\mathcal{M}_{\text{del}} \leftarrow \{i \in \mathcal{M}_{\text{new}} | N_{\text{del}}^i = 1\}$
- 2: Set $\hat{x} \leftarrow x'_t$
- 3: $\forall i \in \mathcal{U}$, set $\hat{x}^i \leftarrow \arg \max_w K^{\theta, i}(w | y_t)$
- 4: $x_{t+\tau} \leftarrow \text{Remove}(\hat{x}, \mathcal{M}_{\text{del}})$
- 5: $\hat{K} \leftarrow \text{Remove}(K^{\theta}(\cdot | y_t), \mathcal{M}_{\text{del}})$
- 6: **return** $x_{t+\tau}, \hat{K}$

τ -leaping rate, $N_{\text{in}}^i \sim \text{Poisson}(\frac{\lambda\tau}{1-t} g^{\theta, i}(x_t))$, so $\lambda = 1$ recovers the standard sampler of Section 2.3 and $\lambda > 1$ makes every step more aggressive. The same counts expand two sequences (line 4): $x'_t = \text{Expand}(x_t, N_{\text{in}})$ is the sequence being edited and still contains the masks of x_t , while the proposal $y_t = \text{Expand}(\tilde{x}_t, N_{\text{in}})$ has masks only at the newly inserted cells. Note that the greedy fill is not written into x'_t , it only serves as a proposal for the corrector. Then the corrector (Algorithm 3) performs a second forward pass, on y_t , and uses $K^{\theta}(\cdot | y_t)$ to substitute incorrect tokens, delete unnecessary newly inserted masks, and provide the unmasking distributions for all remaining masks. For substitutions (line 3), we use argmax over the vocabulary items from $K^{\theta}(\cdot | y_t)$ at the positions $\mathcal{U} = \{i | x_t'^i \neq [\text{M}]\}$, i.e., only at tokens that were already unmasked in x_t . For deletions (line 1), we consider only the newly inserted masks $\mathcal{M}_{\text{new}} = \{i | y_t^i = [\text{M}]\}$, 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 (lines 4–5). Masks inherited from x_t are neither substituted nor deleted. The output of the correction is the new sequence $x_{t+\tau}$ and the updated token probabilities $\hat{K} = \text{Remove}(K^{\theta}(\cdot | y_t), \mathcal{M}_{\text{del}})$. The unmasking of all remaining masks in $x_{t+\tau}$, both inherited and newly inserted, is then performed based on $\hat{K}$ 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, $\text{Expand}(K^{\theta}(\cdot | x_t), N_{\text{in}})$ as shown in line 9 of Algorithm 2. 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 forward passes. Unless stated otherwise, we set *apply_correction* to true at every step, i.e., one correction step follows each proposal step, giving $T/2$ proposal and $T/2$ correction steps. Note that since the model is time-blind, the network never sees t . It enters only through the $\frac{1}{1-t}$ factor in the Poisson rates for insertions, deletions, and unmasking (SampleUnmask), and through the same factor in the training losses.

4 EXPERIMENTS

Setup All models use a transformer backbone with rotary positional embeddings, pre-layernorm and are trained from scratch.² For masked diffusion we use the time-agnostic version with unweighted cross-entropy loss (Zheng et al., 2025) 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 time-blind and uses adaptive layer normalization conditioned on a time embedding. In order to obtain time-blind δ MDM, we pin the time embedding to a constant value of 1, but keep the same architecture as FlexMDM.³ Since δ MDM 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. At inference, unless stated otherwise, δ MDM applies one correction step after every proposal step with both steps counted towards the sampling budget T . We conduct comprehensive evaluation on three different types of tasks: controlled complexity synthetic planning, high-diversity

²We use the transformer implementation provided in Patel et al. (2025a). See Appendix B for more details.

³Removing time from the model’s input has only a small impact even for FlexMDM. See Appendix D.2 for more details.

Model	Gen. PPL ($\downarrow$)						Entropy ($\uparrow$)					
	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
DILM-S [†]	-	-	36.10	29.91	23.20	21.21	-	-	3.45	3.61	3.71	3.79
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
δ MDM	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 T (network forward passes). δ MDM uses insertion scale $\lambda = 0.90$ and one correction step per proposal step from the start, both counted in T . [†]ILM (Patel et al., 2025c) and DILM-S (Patel et al., 2026a), replicated from the implementation and checkpoints provided by Patel et al. (2026a).

unconditional text generation, and low-diversity 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 δ MDM 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). δ MDM is trained for 400k steps with global batch size of 512, while MDM and FlexMDM are trained for 800k steps. We additionally compare against the insertion language models ILM (Patel et al., 2025c) and DILM-S (Patel et al., 2026a), trained on OWT for 800k steps, which we evaluate using the implementation and checkpoints provided by Patel et al. (2026a). 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. For δ MDM, we use the insertion scale $\lambda = 0.90$ in Table 1.

As shown in Table 1, δ MDM achieves lower generative perplexity compared to FlexMDM at all budgets except for $T = 32$. Interestingly, we observe that δ MDM keeps improving in quality at a higher rate compared to FlexMDM as the inference budget increases (see Fig. 1 bottom right). Specifically, at $T = 1024$, δ MDM halves the generative perplexity of FlexMDM (20.35 vs. 40.37) and reduces that of MDM by 47% (20.35 vs. 38.16). Conversely, δ MDM at 128 sampling steps obtains lower Gen. PPL than both FlexMDM and MDM at 1024 steps, an 8x reduction in sampling steps. δ MDM also obtains lower Gen. PPL than the insertion language models ILM (Patel et al., 2025c) and DILM-S (Patel et al., 2026a) at every budget they report, while keeping a much higher token entropy than DILM-S (4.60–4.72 vs. 3.45–3.79). The token entropy for δ MDM is generally lower than that of FlexMDM but it remains stable across all budgets.⁴ Cross-budget plots of token entropy against generative perplexity are in Appendix D.3. Figure 5 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.

4.2 CONDITIONAL GENERATION ON TINYGSM

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 is 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.

⁴Note that the token entropies ranges are lower than the ones reported in the literature on MDMs because we train on variable length sequences which results in lower average entropy.

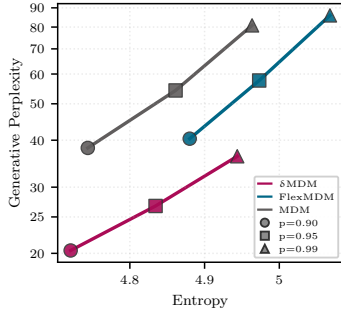


Figure 4: Entropy vs. generation perplexity (GPT-2 Large) on OWT at $T=1024$ for nucleus sampling with $p \in \{0.9, 0.95, 0.99\}$.

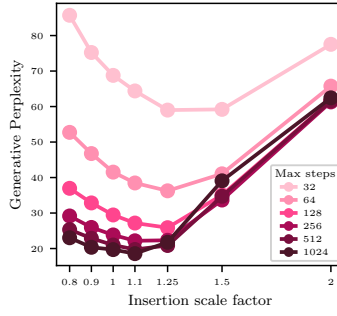


Figure 5: Generation perplexity (GPT-2 Large) of δ MDM on OWT vs. insertion scale λ for different sampling budgets T .

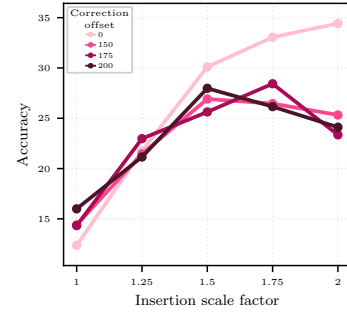


Figure 6: GSM8K pass@1 of δ MDM vs. insertion scale λ at $T=250$ (other budgets in Appendix D.5).

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. For δ MDM, we use the insertion scale $\lambda = 1.75$. As seen in Table 2, δ MDM outperforms FlexMDM at every budget, with the largest gap at low budgets, where δ MDM matches FlexMDM’s accuracy with half the steps. MDM is the strongest model at the lowest budgets ($T \leq 250$), but its accuracy plateaus around 40–41%, and δ MDM overtakes it from $T = 500$ onward.

Model	GSM8K pass@1 (% , $\uparrow$)				
	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
δ MDM	20.62	33.06	42.23	46.85	47.01

Table 2: GSM8K test pass@1 vs. budget T (forward passes). MDM and FlexMDM are trained for 800k steps and δ MDM for 400k. δ MDM uses $\lambda = 1.75$ and one correction step per proposal step from the start, both counted in T .

4.3 ANALYSIS

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

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. 11 and 12, 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. Towards the end, most of the sequence is already fixed, and errors tend to show up as excessive addition of new masks.

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. Note that $\lambda > 1$ is not an extrapolation beyond training: λ scales the per-step leap, and for all $\lambda \leq 2$ the fraction of the predicted gap inserted per step stays within the range $(0, 1)$ seen during training (Section 3.3). As shown in Fig. 5, 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 $\lambda=1.25$ and stays flat at $\lambda=1.5$, 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, which is 0 in all other experiments (Fig. 6, and Appendix D.5 for $T \in \{500, 1000\}$). When correction is applied from the start (offset 0), accuracy increases with λ up to $\lambda=2$ at $T=250$ and up to $\lambda=1.75$ at $T \in \{500, 1000\}$. In contrast, with a late correction start, accuracy peaks at $\lambda=1.5$ 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 $\lambda=1$ 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 the correction step on predictive entropy? The correction step performs a second forward pass over the sequence after the aggressive proposal. We plot the average predictive entropy at the positions that are masked in x_t , comparing $K^\theta(\cdot | x_t)$ from the proposal pass with $K^\theta(\cdot | y_t)$ from the correction pass, where these positions hold the greedy proposal tokens (Fig. 9 in Appendix D.1), and find that entropy is generally lower after the correction forward pass. Since the greedy fill places a concrete token at each of these positions, the proposal tends to favor some of the plausible alignments over others, and conditioning on it contributes to reducing the alignment ambiguity that inflates FlexMDM’s entropy at mask slots (Fig. 8). We hypothesize that these sharper predictions are what allow δ MDM to unmask more tokens per step, consistent with the budget curves in Tables 1 and 2.

5 RELATED WORK

There have been several 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., 2026b) 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 could address this, but it would considerably more complex. 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 AND OUTLOOK

This work introduces δ MDM, 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. δ MDM Pareto-dominates FlexMDM on the speed-quality trade-off, exhibits better scaling with inference budget across unconditional as well as conditional text generation tasks. More broadly, our results hint at a division of labor. The discrete diffusion framework, with its fixed noising process and simple training objective, provides a stable recipe for training from scratch. However, the gains of δ MDM come from departing from it in two ways: the model is time-blind, and it is trained on corruptions produced by its own proposals. Together, these make each sampling step behave less like integrating a prescribed continuous-time process and more like repeatedly applying a single learned refinement operator. This suggests that adapting pretrained diffusion models with such self-correction objectives, in the spirit of fixed-point iteration, could be a promising route to faster and higher-quality generation.

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>.
- BumJun Kim, Dongjae Jeon, Dueun Kim, Wonje Jeung, and Albert No. Rainbow padding: Mitigating early termination in instruction-tuned diffusion llms. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 18977–19001, 2026a. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/203221ce5041eddb0800cef81bf987-Paper-Conference.pdf.
- Jaeyeon Kim, Lee Cheuk-Kit, Carles Domingo-Enrich, Yilun Du, Sham Kakade, Timothy Ngatiaoco, 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, 2026b. URL <https://arxiv.org/abs/2607.26504>.
- Jinsong Li, Xiaoyi Dong, Yuhang Zang, Yuhang Cao, Jiaqi Wang, and Dahua Lin. Beyond fixed: Training-free variable-length denoising for diffusion large language models. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 91715–91731, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/940380c12da75e64351cceff4c880557-Paper-Conference.pdf.

- 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>.
- Hengchang Liu, Zhao Yang, and Bing Su. Diffusion lms can approximate optimal infilling lengths implicitly, 2026a. URL <https://arxiv.org/abs/2602.00476>.
- Liming Liu, Binxuan Huang, Zixuan Zhang, Xin Liu, Bing Yin, and Tuo Zhao. Backplay: Head-only look-back self-correction for diffusion language models, 2026b. 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.
- Dhruv 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>.
- Dhruv 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>.
- Dhruv 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>.
- Dhruv 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>.
- Dhruv 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 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. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 130136–130184. Curran Associates, Inc., 2024. doi: 10.52202/079017-4135. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/eb0b13cc515724ab8015bc978fdde0ad-Paper-Conference.pdf.
- Subham Sekhar Sahoo, Jean-Marie Lemerrier, 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. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 103131–103167. Curran Associates, Inc., 2024. doi: 10.52202/079017-3277. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/bad233b9849f019ae5e5cc60cef70f-Paper-Conference.pdf.

- 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>.
- Yuxuan Song, Zheng Zhang, Cheng Luo, Pengyang Gao, Fan Xia, Hao Luo, Zheng Li, Yuehang Yang, Hongli Yu, Xingwei Qu, Yuwei Fu, Jing Su, Ge Zhang, Wenhao Huang, Mingxuan Wang, Lin Yan, Xiaoying Jia, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Yonghui Wu, and Hao Zhou. Seed Diffusion: A Large-Scale Diffusion Language Model with High-Speed Inference, August 2025. URL <http://arxiv.org/abs/2508.02193>.
- 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>.
- Haobo Xu, Sirui Chen, Yuanchen Bei, Lingjie Chen, Yuchen Yan, Dongqi Fu, Jingrui He, and Hanghang Tong. Predict, don’t iterate: Efficient adaptive-length infilling for diffusion language models, 2026. URL <https://arxiv.org/abs/2609.02108>.
- Andrew Zhang, Anushka Sivakumar, Chia-Wei Tang, and Chris Thomas. Flexible-length text infilling for discrete diffusion models. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 31344–31359, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.1597. URL <https://aclanthology.org/2025.emnlp-main.1597/>.
- 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 Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 63186–63227, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/9e3b203e72c4e058de26d02a92a81844-Paper-Conference.pdf.

APPENDIX

A Summary of Notation	14
B Experimental Details	16
C Additional Tasks	16
C.1 Synthetic Planning: Star Graphs	16
D Additional Analysis	17
D.1 Multiple alignment problem	17
D.2 Time-blindness	18
D.3 Entropy vs. generation perplexity across sampling budgets	18
D.4 How are corrections distributed?	18
D.5 GSM8K accuracy vs. insertion scale	20
D.6 Length of generated sequences on OWT	20
E Extended Related Work	20
F Randomly Sampled Generations	21
F.1 OpenWebText	21
F.2 GSM8K	23

A SUMMARY OF NOTATION

Table 3 summarizes the notation used throughout the paper.

Notation	Description
Sequences and spaces	
$\llbracket n \rrbracket$	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, $\mathbb{Z} = \bigcup_{k \in \llbracket L \rrbracket} \mathbb{V}^k \times \mathbb{S}_k$, where $\mathbb{S}_k := \{(s^1, \dots, s^k) \in \llbracket L \rrbracket^k \mid s^1 < \dots < s^k\}$
$\bar{\mathbb{Z}}, f_{\text{contract}}$	$\bar{\mathbb{Z}} = (\mathbb{V} \cup \{\text{[D]}\})^L$, and the isomorphism $f_{\text{contract}} : \bar{\mathbb{Z}} \rightarrow \mathbb{Z}$ 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
$T_{\text{in}}^i, T_{\text{um}}^i$	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 $\mathbb{V} \cup \{\text{[DEL]}\}$
$g^{\theta, i}(x)$	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
ρ	Training insertion fraction, $\rho \sim \text{Uniform}(0, 1)$; the fraction of the predicted remaining gap inserted in one step
λ	Inference insertion scale multiplying the τ -leaping insertion rate; each step inserts the fraction $\rho_t = \frac{\lambda \tau}{1 - t} \leq \lambda/2$
$\text{Expand}(\cdot, N_{\text{in}}), \pi(i)$	Inserts masks into the gaps, shifting later cells to the right; $\pi(i)$ is the new index of cell i (Section 3.1)
$\text{Remove}(\cdot, D)$	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, $x'_t = \text{Expand}(x_t, N_{\text{in}})$
y_t	Proposal given to the corrector, $y_t = \text{Expand}(\tilde{x}_t, N_{\text{in}})$
q_t	Alignment of y_t to x_1 , $q_t = \text{Expand}(s_t, N_{\text{in}})$; $q_t^i = -1$ 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
$0 = t_0 < \dots < t_N = 1, \tau$	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. δ MDM is trained for 400k steps, while MDM and FlexMDM are trained for 800k steps. ILM (Patel et al., 2025c) and DILM-S (Patel et al., 2026a) are evaluated from the implementation and 800k step checkpoints provided by Patel et al. (2026a). OpenWebText sequences are padded to length 1024, and TinyGSM sequences use length 512. For the main results (Tables 1 and 2), δ MDM uses the insertion scale $\lambda = 0.90$ on OpenWebText and $\lambda = 1.75$ on TinyGSM. Correction is applied after every proposal step from the first step (offset 0), and both proposal and correction steps count towards the budget T .

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	$\approx 87\text{M}$
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	Star _{easy} ($\uparrow$)	Star _{medium} ($\uparrow$)	Star _{hard} ($\uparrow$)
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.

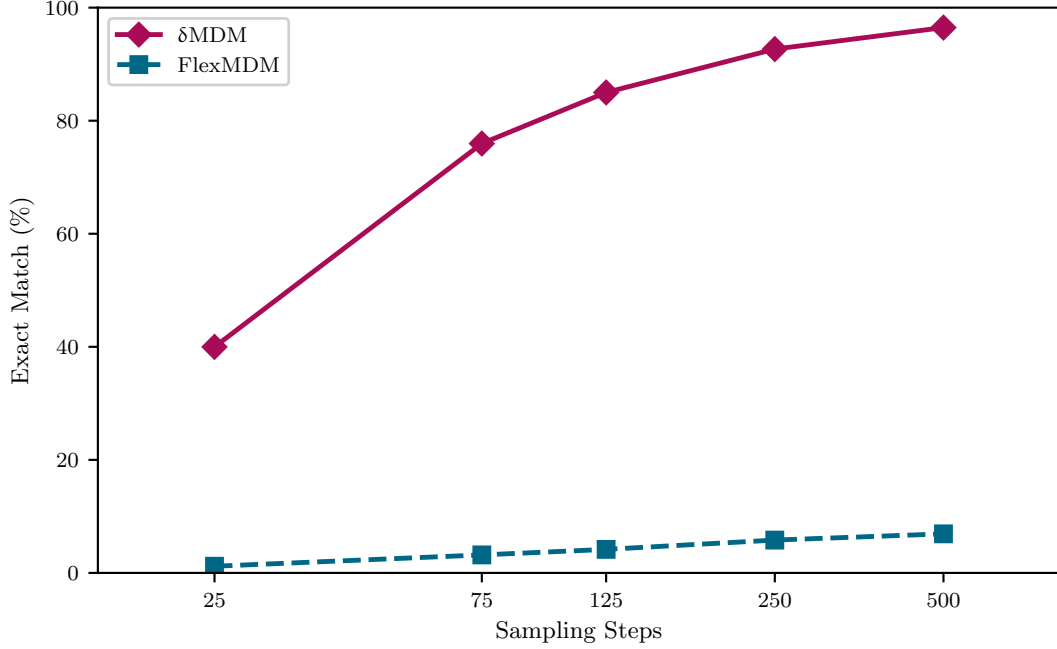


Figure 7: $\text{Star}_{\text{hard}}$ exact match ($\uparrow$) vs. sampling budget for δ MDM and FlexMDM at $T \in \{32, 64, 128, 256, 512\}$.

D ADDITIONAL ANALYSIS

D.1 MULTIPLE ALIGNMENT PROBLEM

Figure 8 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 9 shows the same quantity for δ MDM at sampling budget $T=512$, before and after the correction step.

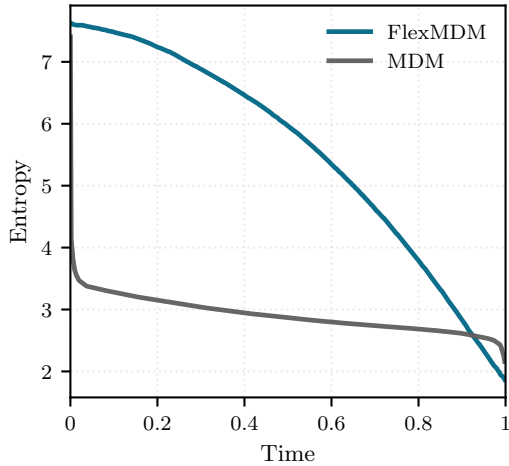


Figure 8: Entropy of the predicted token distribution at masked positions vs. time t on OWT for FlexMDM and MDM.

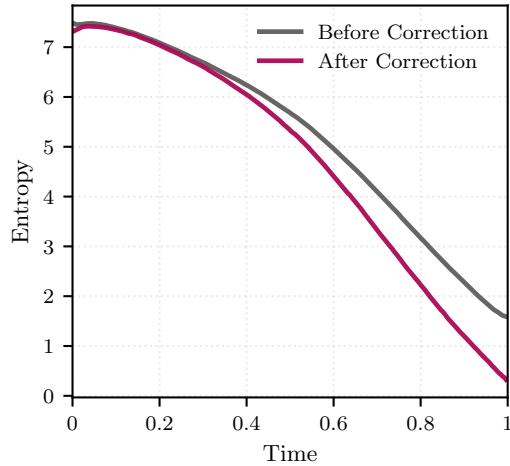


Figure 9: Entropy of the predicted token distribution at masked positions vs. time t on OWT for δ MDM at $T=512$, before and after correction.

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 the model’s input makes only a small difference in performance: the gap is within 0.25 points at $T \leq 500$ and about 2.3 points at $T = 1000$.

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

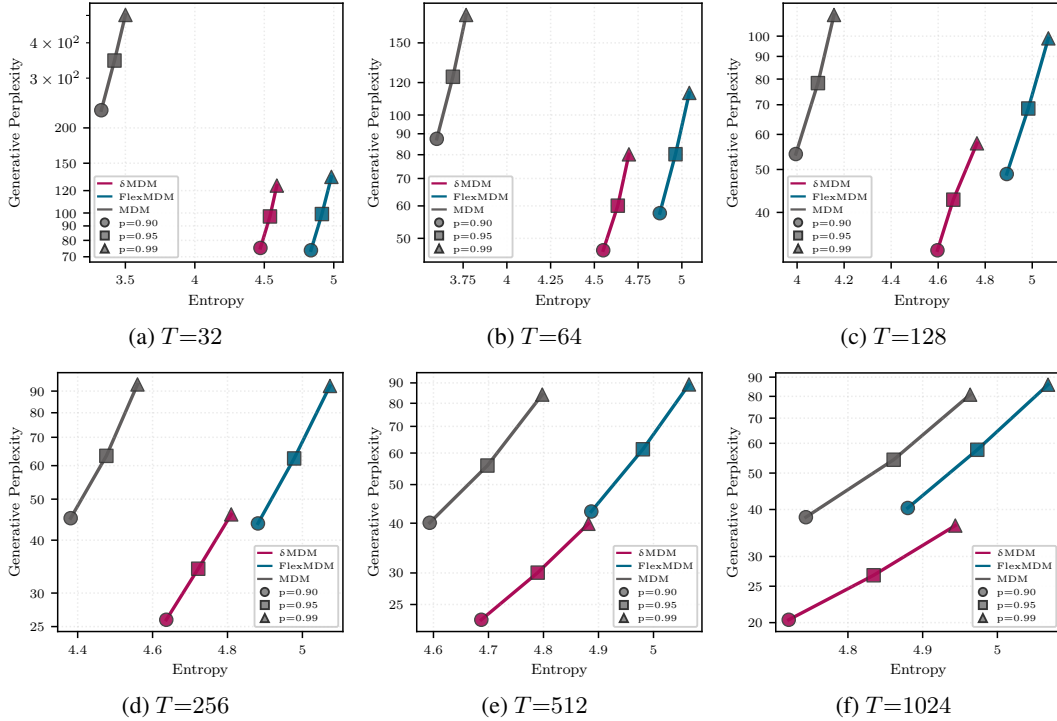


Figure 10: Entropy vs. generation perplexity (GPT-2 Large) on OWT for nucleus sampling with $p \in \{0.9, 0.95, 0.99, 1\}$ at the remaining sampling budgets.

D.4 HOW ARE CORRECTIONS DISTRIBUTED?

Figures 11 and 12 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.

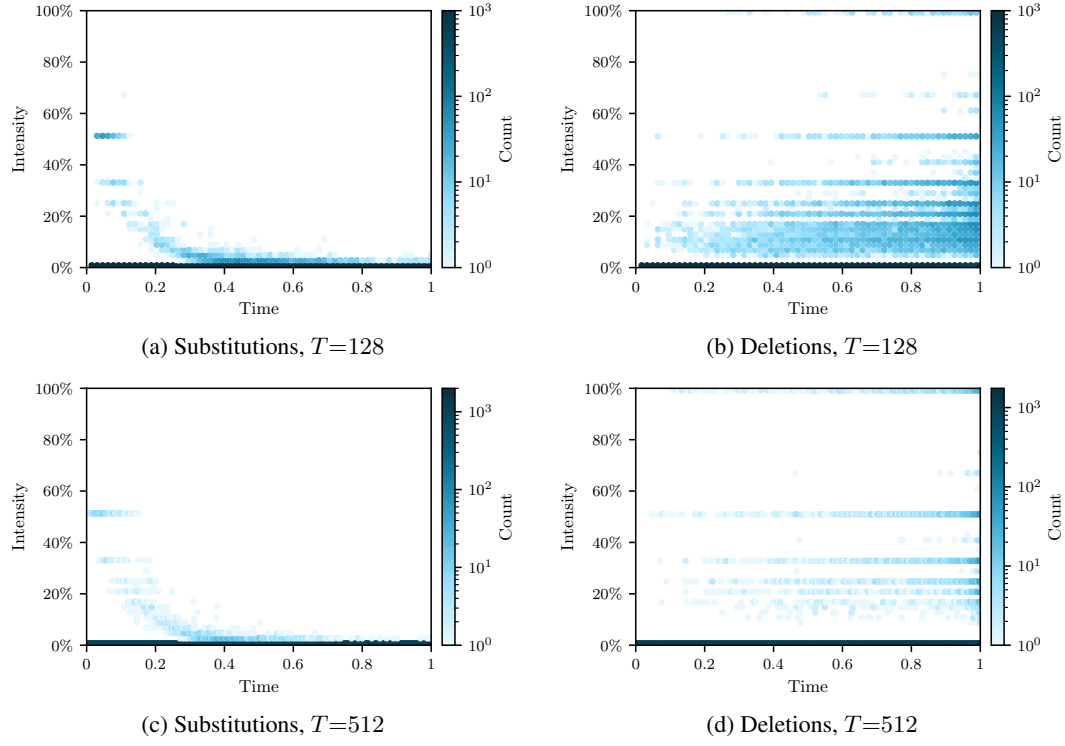


Figure 11: Distribution of substitution and deletion intensity over normalized time for δ MDM on OWT.

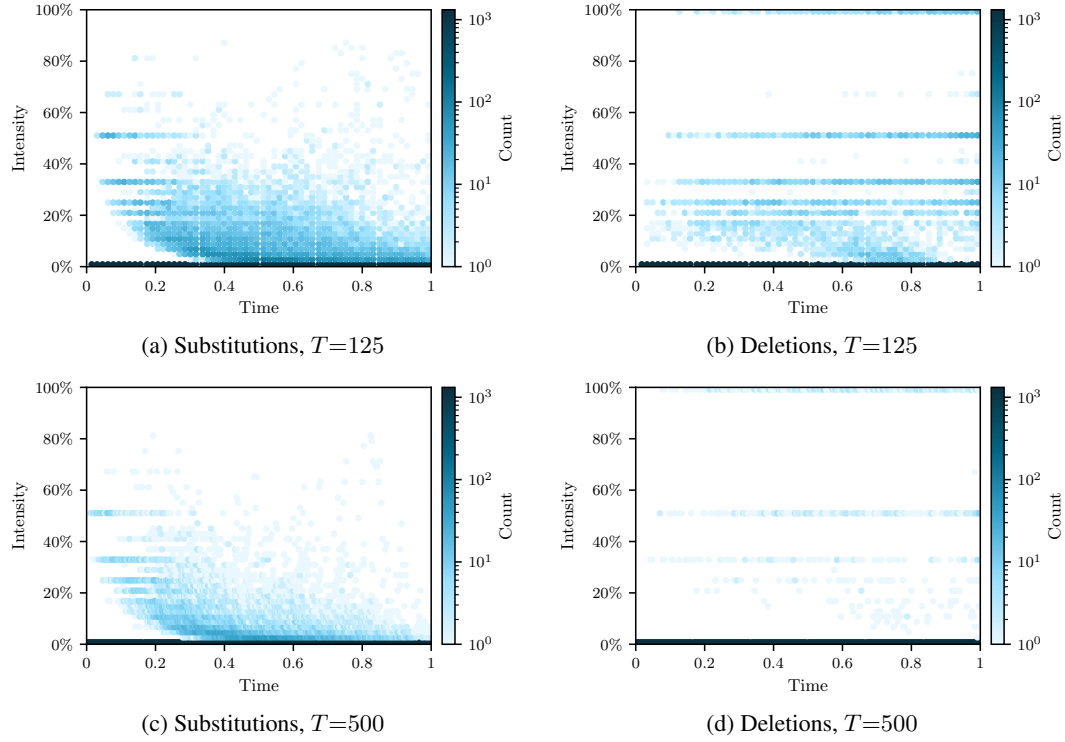


Figure 12: Distribution of substitution and deletion intensity over normalized time for δ MDM on GSM8K.

D.5 GSM8K ACCURACY VS. INSERTION SCALE

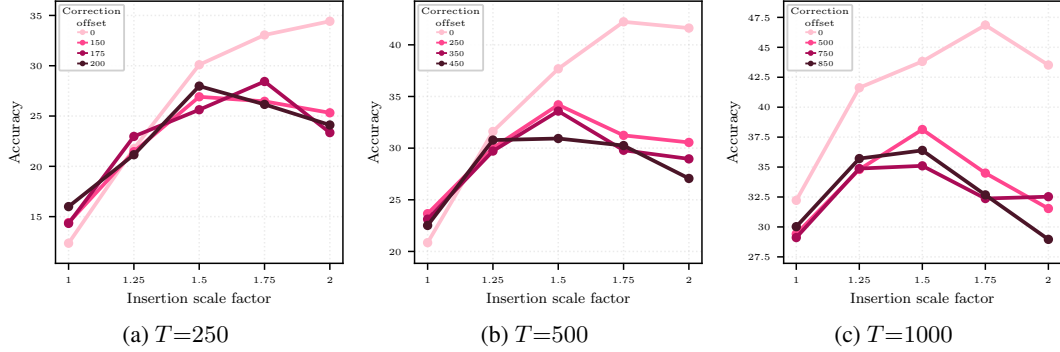


Figure 13: GSM8K pass@1 vs. insertion scale at sampling budgets $T \in \{250, 500, 1000\}$.

D.6 LENGTH OF GENERATED SEQUENCES ON OWT

Figure 14 compares the quantiles of the generated sequence lengths against the OWT validation set, and Fig. 15 shows how the mean length of δ MDM generations changes with the insertion scale λ .

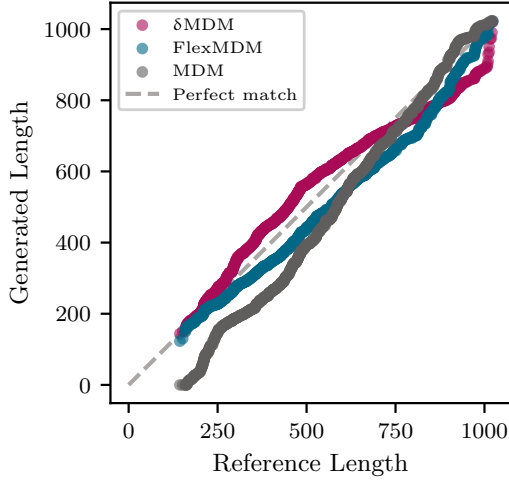


Figure 14: Q-Q plot of the sequence length quantiles of δ MDM, FlexMDM, and MDM generations vs. OWT validation.

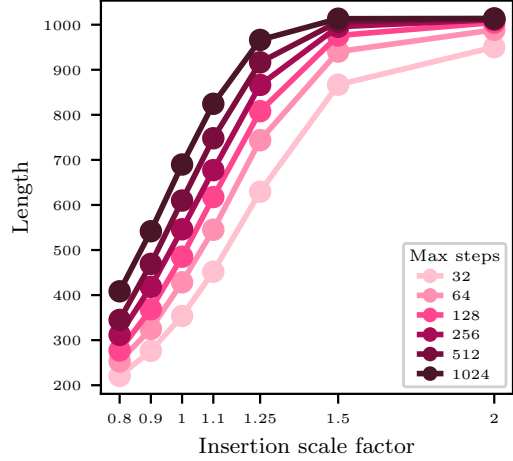


Figure 15: Mean length of δ MDM generations on OWT vs. insertion scale λ for different sampling budgets T .

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., 2026b) 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. Seed Diffusion (Song et al., 2025) augments MDM training with an edit-based objective that teaches the model token-level edits by recovering the clean sequence from a noisy version that is generated by a controlled corruption process. It drops carry-over unmasking, so the model re-evaluates already-unmasked tokens and can implicitly self-correct.

Variable-Length Masked Diffusion: Fixed-canvas MDMs achieve variable-length control by filling extra positions with padding tokens. Rainbow Padding Kim et al. (2026a) improves on this by using a cyclic sequence of distinct padding tokens to avoid premature truncation. DAEDAL Li et al. (2026) instead iteratively inserts mask tokens at the end of the initial canvas based on EOS confidence, adapting the canvas length to the prompt before denoising, and further expands low-confidence masked regions with additional mask tokens for dynamic length control during inference. In contrast, insertion-based diffusion approaches such as FlexMDM Kim et al. (2025a) start from an empty canvas and grow the sequence by iteratively inserting new masks at arbitrary positions while unmasking previously inserted ones. Several flexible-length infilling methods predict the optimal infilling length before denoising, such as using initial-step denoising confidence Liu et al. (2026a) or a lightweight trained length-probing head Xu et al. (2026). Zhang et al. (2025) takes an orthogonal approach, allowing token positions to evolve during denoising while using optimal-transport position coupling to preserve relative order, which enables flexible-length text infilling without explicit insertion or deletion operations. Although most prior work on correction focuses on fixed-length MDMs, few approaches address self-correction in variable-length settings like DreamOn (Wu et al., 2026), which controls length through expansion and deletion operations that adaptively insert and remove masks, achieving superior performance on code-infilling benchmarks. However, it does not perform substitutions, so errors from carry-over unmasking persist.

F RANDOMLY SAMPLED GENERATIONS

F.1 OPENWEBTEXT

T=128

Google’s new Google Now series is coming. (Photo: YouTube)

Google is now expanding its Android. It also plans to help it deliver its 4K Chromecast, which allows users to stream their videos from anywhere in their homes, as they do social media like Netflix. Google said that it has a wide set of messaging features and integration with Android. The company also said communication features allow conversations between YouTube and users to up to 18 percent in short-range videos and up to 90 percent. Android users said the Android Chromecast works well with Netflix, while holding back on user engagement when compared to Netflix’s.

The Chromecast, which is available to all Android device makers as part of its new cross-platform standards, was released with Android 2.0, as well as support for iPhone and iPad devices as well. It will be made available soon.

T=256

Please enable Javascript to watch this

COLUMBUS, Ohio — The Columbus Crew today announced the signing of 21-year-old international forward Jermain Zargent and a number of players.

21-year-old Zargent, MLS’s second overall pick in the 2014 SuperDraft and was selected in the 2015 MLS Draft, joined by the Columbus Crew for training on Wednesday.

The first time Zargent has come to Columbus since he made it to the United States national team last summer to join the New York Red Bulls.

Zargent is the second-ever player named to the National Team squad (17 caps) this season. He was the 13th International and returned to the team after a video of him that went viral.

Speaking to The Columbus Pioneer Press, Zargent said: "I came to the team and let me play in the summer. This is the first year I've found my opportunity to play in Columbus and I am excited to have opportunity and play with a good team."

Zargent will join the Red Bulls and the United States' first-ever head coach, head coachumberto Gonzalez.

He had a few words: "I know what my first year is to be, and training in the name of. I do a great job of working, and think about hard environment."

native Zargent has also played for Germany and some medals at the European Championships. Zargent is happy to be out with the Red Bulls and in Columbus.

"I'm excited to play for the team, and I'm going to to Columbus. is a great opportunity for my teammates and fans and it's a big part of my game this year," Zargent. "It's a great year and that's why I put myself in Columbus. I'm going to play with myself. I love this team, my teammates and coaches and myself. I'm blessed and I'll be a part of this team this year."

Thank you to The Columbus Journal Sentinel for continuing to reach out to Zargent and his New Red Bulls teammates.

T=512

A former city worker is in the jail and a local who was convicted of his crimes is now off the job and a \$1 million fine.

A Cook County district judge yesterday placed thei on probation, according to the Daily News. Court documents show that Esther Laili, 27, was convicted on first-degree burglary, criminal mischief, attempted grand larceny, obstruction of public service, forgery, and unlawful use of a firearm. Laili was already on his probation when he was arrested and released from jail.

"I just woke up," said Laili and his wife. "It's a sad day for the city's entire community."

The city also claims the former worker has been arrested and and is now off the job. City officials say they're being contacted by police about Laili's record. He's also threatened with the release of his wife, according to court documents. If they say, "with support, shock," and "belief, fear and harm," Laili is going to have to pay the rest of his sentence.

Last year, the county stepped in to the McDonald McDonald case, and then even revoked the city's five-year probation. For Laili, the judge cast a reasonable doubt that the county would maintain his probation instead of his five-year prison sentence, the Daily News reports.

T=1024

Federal authorities have responded to an Oklahoma oil spill 150 miles north of Michalaw, N.C., and containment units in several areas close to the site. The structures at the Hoover Dam are more than three times the size and have been working to contain the entire oil spill, officials said.

Andrew Coles, a project operations supervisor for the agency, said authorities immediately responded to the emergency response and oil spill.

"We have personnel at the center, and we hope can assist them," Coles said.

If containment units fail, the damage could be catastrophic or be significant.

“We expect our response team to work not only for those areas but for the natural and environment and public safety,” said Department of Natural Resources spokesman Robert Reyna.

The structures at the Hoover Dam — the unit, containment tube, and chemical reinforcement ducts — are hazardous to about 500 feet of water, officials said.

“While Hoover Dam is safe, this spill is dangerous and has the potential to cause damage to the natural and environment,” Reyna said in a statement.

Rivera officials said a decision was made to allow the response team to work with those responsible to restore safety to public health, but did not comment on the cost or impact of the cleanup or the impact of the spill.

“This cleanup is ongoing as is always planned,” Department of Natural Resources Reyna said. “The full cleanup of the spill will be carried out in any major waste facility.”

Just as containment equipment works at water tanks behind the shores of the Great Lakes, destructive dispersants at the tanks can be taken, and the water can be released from smog.

The Sept. 11 earthquake destroyed the oil industry’s largest operating power plant, and killed more than 230,000 people in Oklahoma alone, according to the Department of Energy and Environment.

F.2 GSM8K

T=125

Question

To be promoted to the next school year, Jane takes 3 tests that together must total at least 42 points. On her first test, Jane scored 15 points, on the second test she scored 18 points. What is the minimum number of points she must score on the third test to pass?

```
def simple_math_problem() -> int:
    """
```

To be promoted to the next next school year, Jane takes 33 that together must total at least 42 points
On her first test, Jane scored 15 points, on the second test she scored 18 points.
What is the minimum number of points she must score on the third test to pass?

```
    """
    minimum_total_score = 42
    first_test_score = 15
    second_test_score = 18
    third_test_score = minimum_total_score - (first_test_score + second_test_score)
    result = third_test_score
    return result
```

T=250

Question

Samwell owns an appliances store. For this week, one-fourth of their sales are smart TVs, one-eighth are analog TVs, and the rest are OLED TVs. If they were able to sell a total of 40 TVs, how many OLED TVs were they able to sell?

```
def simple_math_problem() -> int:
    """
```

Samwell owns an appliances store.

For this week, one-fourth of their sales are smart TVs, one-eighth are analog TVs, and the rest are OLED TVs.

If they were able to sell a total of 40 TVs, how many OLED TVs were they able to sell?

```
'''
total_tvs = 40
smart_tvs = total_tvs * 1/4
analog_tvs = total_tvs * 1/8
OLED_tvs = total_tvs - smart_tvs - analog_tvs
result = OLED_tvs
return result
```

T=500

Question

Brian's basement was damp and musty, so he bought a dehumidifier to remove moisture out of the air. The device has three speeds: low, medium, and high. Brian tested the device's efficiency and he found that the low setting removes 1 liter of water out of the air per day, the medium setting removes twice as much water per day as the low setting, and the high setting removes twice as much water per day as the medium setting. If Brian ran the dehumidifier for 3 days on the low setting, then an additional 3 days on the medium setting, and then an additional 5 days on the high setting, what is the total amount of water that the dehumidifier removed from the air in his basement, in liters?

```
def simple_math_problem() -> int:
'''
    Brian's basement was damp and musty, so he bought a dehumidifier to remove moisture out of the air.
    The device has three speeds: low, medium, and high.
    Brian tested the device's efficiency and he found that the low setting removes 1 liter of water out of
    the air per day, the medium setting removes twice as much water per day as the low setting, and the high
    setting removes twice as much water per day as the medium setting.
    If Brian ran the dehumidifier for 3 days on the low setting, then an additional 3 days on the medium
    setting, and then an additional 5 days on the high setting, what is the total amount of water that the
    dehumidifier removed from the air in his basement, in liters?
'''
    low_setting = 1
    medium_setting = 2 * low_setting
    high_setting = 2 * medium_setting

    low_setting_days = 3
    medium_setting_days = 3
    high_setting_days = 5

    low_removed = low_setting * low_setting_days
    medium_removed = medium_setting * medium_setting_days
    high_removed = high_setting * high_setting_days

    total_water_removed = low_removed + medium_removed + high_removed

    result = total_water_removed
    return result
```

T=1000

Question

In a dance class of 20 students, 20% enrolled in contemporary dance, 25% of the remaining enrolled in jazz dance, and the rest enrolled in hip-hop dance. What percentage of the entire students enrolled in hip-hop dance?

```
def simple_math_problem() -> int:
'''
    In a dance class of 20 students, 20%
```

```

What percentage of the entire students enrolled in hip-hop dance?
'''
total_students = 20
contemporary_percentage = 20
contemporary_students = (contemporary_percentage/100) * total_students
remaining_students = total_students - contemporary_students
jazz_percentage = 25
jazz_students = (jazz_percentage/100) * remaining_students
hip_hop_students = remaining_students - jazz_students
percentage = (hip_hop_students / total_students) * 100
result = percentage
return result

```

Below we show the substitutions from three intermediate steps of a GSM8K generation trajectory with a total budget of $T=250$ steps, followed by the final completion. Mask tokens are denoted by [M], and **orange** highlighted transitions mark the substitutions that occurred at that time step.

Question

Marilyn wants to buy a pair of boots online. On Amazon.com, the boots cost \$16 and shipping costs \$4. On eBay, those same pair of shoes are only \$13, but shipping costs twice as much as it does on Amazon. How much more expensive do the boots work out to be on eBay?

$t=146$

```

def[M][M][M][M][M][M] Marilyn[M] to[M][M][M][M] online.
[M] Amazon the boots[M] $1[M][M] shipping costs[M][M][M][M][M],[M][M][M] only $[M][M][M][M]
as[M][M] does on Amazon[M][M][M] more expensive[M] the[M] work out[M] be[M][M]
[M][M]n[M][M][M][M]
[M] amazon_shipping = [M][M][M][M] amazon[M][M][M][M][M][M][M][M][M][M][M][M][M] =
amazon_shipping_cost[M] [M][M][M][M][M]n_boots + amazon[M][M][M][M][M][M][M][M] +[M][M][M]
[M] [= → ebay] _total - _total_cost[M][M][M][M] return result

```

$t=148$

```

def[M][M][M] int[M][M][M] Marilyn[M] to[M][M][M][M] online.
[M] Amazon the boots[M] $1[M][M] shipping costs[M][M][M][M][M],[M][M][M] only $[M][M][M][M]
as[M][M] does on Amazon[M][M][M] more expensive[M] the[M] work out[M] be[M][M] [M]
n[M][M][M][M]
[M] amazon_shipping = [M][M][M][M] [amazon → ebay] [M][M][M][M][M][M][M][M][M][M][M][M]
= amazon_shipping_cost[M] [M][M][M][M][M]n_boots + amazon[M][M][M][M][M][M][M][M]
+[M][M][M] [M] ebay_total - _total_cost[M][M][M][M] return result

```

$t=152$

```

def[M][M][M] int[M][M][M] Marilyn[M] to[M][M][M][M] online.
[M] Amazon[M] the boots[M] $1[M][M] shipping costs[M][M][M] On[M],[M][M][M] only $[M][M][M]
twice as[M][M] does on Amazon[M][M][M] more expensive[M] the[M] work out[M] be eBay[M] [M]
n[M][M][M][M]
[M] amazon_shipping = [M][M][M] ebay[M][M][M][M][M][M][M][M] ebay[M][M][M] [= → *]
amazon_shipping_cost[M] [M][M][M][M][M]n_boots + amazon[M][M] ebay[M][M][M]oes +[M][M][M]
[M] ebay_total - _total_cost[M][M][M][M] return result

```

Final

```
def simple_math_problem() -> int:
    """
    Marilyn wants to buy a pair of boots online.
    On Amazon.com, the boots cost $16 and shipping costs $4.
    On eBay, those same pair of shoes are only $13, but shipping costs twice as much as it does on
    Amazon.
    How much more expensive do the boots work out to be on eBay?
    """
    martn_boots = 16
    amazon_shipping_cost = 4
    ebay_shoes = 13
    ebay_shipping_cost = 2 * amazon_shipping_cost

    martn_total_cost = martn_boots + amazon_shipping_cost
    ebay_total_cost = ebay_shoes + ebay_shipping_cost

    cost_difference = ebay_total_cost - martn_total_cost

    result = cost_difference
    return result
```