FINE-TUNING MASKED DIFFUSION FOR PROVABLE SELF-CORRECTION

Jaeyeon Kim[1⋆] **Seunggeun Kim**[2⋆] **Taekyun Lee**[2⋆]
David Z. Pan[2] **Hyeji Kim**[2] **Sham Kakade**[1,3] **Sitan Chen**[1]
[1]Harvard University [2]UT Austin [3]Kempner Institute

ABSTRACT

A natural desideratum for generative models is *self-correction*–detecting and revising low-quality tokens at inference. While Masked Diffusion Models (MDMs) have emerged as a promising approach for generative modeling in discrete spaces, their capacity for self-correction remains poorly understood. Prior attempts to incorporate self-correction into MDMs either require overhauling MDM architectures/training or rely on imprecise proxies for token quality, limiting their applicability. Motivated by this, we introduce PRISM–**P**lug-in **R**emasking for **I**nference-time **S**elf-correction of **M**asked Diffusions–a lightweight, model-agnostic approach that applies to any pretrained MDM. Theoretically, PRISM defines a self-correction loss that provably learns per-token quality scores, without RL or a verifier. These quality scores are computed in the same forward pass with MDM and used to detect low-quality tokens. Empirically, PRISM advances MDM inference across domains and scales: Sudoku; unconditional text (170M); and code with LLaDA (8B).

1 INTRODUCTION

Masked Diffusion Models (MDMs) (Sahoo et al., 2024; Gat et al., 2024; Shi et al., 2024) have emerged as a promising approach for generative modeling in discrete domains. At inference time, they start from a fully masked sequence and gradually unmask the masked tokens in arbitrary order to obtain a clean sequence. This flexibility in generation order, combined with recent approaches to deployment at scale (Nie et al., 2025; Ye et al., 2025; DeepMind, 2025; Labs et al., 2025; Gong et al., 2025), has enabled them to surpass their autoregressive counterparts on several downstream tasks, such as reasoning, coding, and planning.

However, the standard MDM design lacks the ability to self-correct–*a natural desideratum for generative models*—that is, identifying low-quality or incorrect tokens and correcting them at inference. This limitation is particularly crucial in MDMs due to their parallel, few step inference nature. Parallelized inference with only a few steps makes MDMs more susceptible to dependency errors across positions (Liu et al., 2024a; Xu et al., 2024; Kang et al., 2025), as they model per-position unmasking posteriors rather than the full joint distribution over positions.

Several recent works have attempted to overcome this limitation in MDMs (Campbell et al., 2022; Lezama et al., 2022b; Zhao et al., 2024; Wang et al., 2025; von Rütte et al., 2025; Peng et al., 2025). However, these approaches come with one of two drawbacks: they either **(1)** indirectly or inaccurately learn a notion of token quality and are potentially inefficient to evaluate or **(2)** entirely change the MDM framework or architecture, limiting their applicability.

Contribution. In this work, we propose a simple, principled solution addressing these limitations, **PRISM**: **P**lug-in **R**emasking for **I**nference-time **S**elf-correction of **M**asked diffusions, a plug-and-play fine-tuning framework that easily adapts to any pretrained MDM (addressing **(1)**) and directly learns the clean token's quality given a partially masked sequence (addressing **(2)**).

Theoretically, PRISM introduces a novel self-correction loss that *provably* learns the per-token quality scores. It attaches a lightweight adapter to any pretrained MDM and fine-tunes with this loss; at inference, the learned scores detect low-quality tokens to *remask* and potentially be revised

⋆ Jaeyeon Kim and Seunggeun Kim contributed equally; Taekyun Lee is also a co–first author.

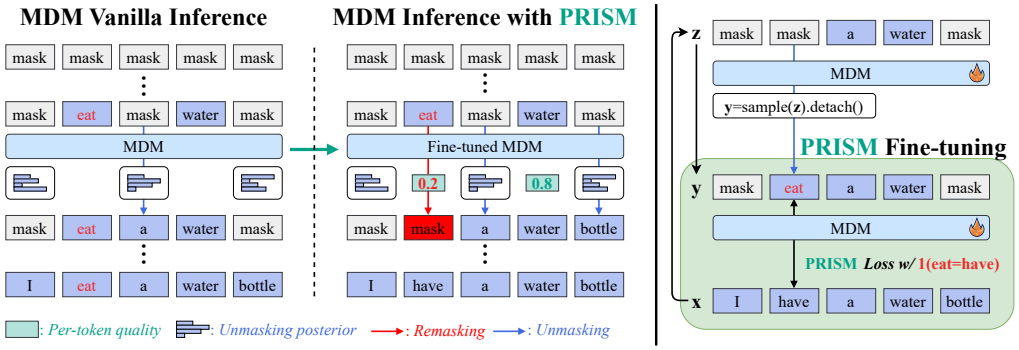

Figure 1: PRISM. (**Left**) MDM learns *unmasking posterior* to unmask tokens, after which they remain fixed. (**Middle**) PRISM fine-tuning introduces *per-token quality*, which is used to detect incorrect tokens and *remask* them. At inference, the fine-tuned MDM jointly computes *unmasking posterior* and *per-token quality*, respectively performing unmasking and remasking. (**Right**) PRISM training pair generation: sample a masked sequence $\mathbf{z}$ from $\mathbf{x}$; from $\mathbf{z}$, unmask a chosen position using the pretrained MDM to obtain $\mathbf{y}$.

in subsequent inference steps (Figure 1, middle). **Practically**, PRISM improves MDM inference across a range of domains and scales: Sudoku, unconditional text generation with a 170M MDM, and code generation with LLaDA (Nie et al., 2025), an open-source 8B MDM. Notably, on LLaDA, with < 500 GPU-hours of fine-tuning, it acquires the ability to self-correct, outperforming baselines on MBPP (Austin et al., 2021b) and HumanEval (Chen, 2021).

2 Preliminaries

2.1 Masked Diffusion Models

Notation. Assume we aim to learn the distribution over sequences of length L with vocabulary $\mathcal{V}$, e.g., natural language, namely the true distribution $\mathbf{x} \sim p_{\text{data}}$. $\mathbf{x}^i$ denotes the i-th element of a given sequence $\mathbf{x} = (\mathbf{x}^1, \ldots, \mathbf{x}^L)$ and $\Delta(\mathcal{V})$ indicates the simplex of probability distributions over $\mathcal{V}$.

MDM training. Although MDMs can be interpreted from several viewpoints, we adopt an *any-order* language model viewpoint (Ou et al., 2024; Zheng et al., 2024), which is essential to understanding our PRISM framework. Roughly speaking, MDM learns the posterior marginals over clean tokens in each masked position, conditioned on a partially masked sequence. We now formalize how this objective is defined and trained. To *learn* the posterior, MDM performs a *masking process* using an auxiliary mask token $\mathbf{m}$. For a given $\mathbf{x} \sim p_{\text{data}}$, this masking process samples a partially masked sequence $\mathbf{z} \in (\mathcal{V} \cup \{\mathbf{m}\})^L$ defined in two equivalent ways (see Appendix B.2 for equivalence):

(1) Draw $n \sim \text{Unif}\{0, \ldots, L\}$ and replace the tokens at uniformly selected n indices in $\mathbf{x}$ with $\mathbf{m}$.
(2) Draw $t \sim \text{Unif}[0, 1]$ and for each i, replace $\mathbf{x}^i \in \mathcal{V}$ with $\mathbf{m}$ with probability t.

To clarify, in (1), for a given n, we uniformly sample an index set from all size-n subsets of $\{1, \ldots, L\}$, and mask corresponding positions of $\mathbf{x}$. This masking process defines a random variable $\mathbf{z}$ conditioned on $\mathbf{x}$, and marginalizing it over $\mathbf{x} \sim p_{\text{data}}$ induces a joint distribution over $(\mathbf{x}, \mathbf{z})$.

We refer to the resulting coordinate-wise posterior conditioned on $\mathbf{z}$ as the *unmasking posterior*; $p(\mathbf{x}^i = \cdot \,|\, \mathbf{z})$, i.e., the distribution over clean tokens at position i given a partially masked sequence $\mathbf{z}$. This unmasking posterior is the core modeling component in MDM and is parameterized by a neural network f_θ, which takes $\mathbf{z}$ as an input and outputs a tensor of shape $|\mathcal{V}| \times L$. Its i-th column, $f_\theta^i(\cdot \,|\, \mathbf{z}) \in \Delta(\mathcal{V})$, models the unmasking posterior $f_\theta^i(v \,|\, \mathbf{z}) \approx p(\mathbf{x}^i = v \,|\, \mathbf{z})$. To train f_θ, MDM minimizes the following loss:

$$\mathcal{L}(\theta) := \mathbb{E}_{\mathbf{x},\mathbf{z}} \left[\frac{1}{n} \sum_{i:\, \mathbf{z}^i = \mathbf{m}} -\log f_\theta^i(\mathbf{x}^i \,|\, \mathbf{z}) \right], \tag{1}$$

which is the cross-entropy loss summed over all masked indices. Here, the expectation is taken over the joint distribution of $(\mathbf{x}, \mathbf{z})$ defined above. As desired, the unique minimizer $f_{\theta^\star}$ of the loss $\mathcal{L}_\theta$

satisfies $f^i_{\theta^\star}(v \mid \mathbf{z}) = p(\mathbf{x}^i = v \mid \mathbf{z})$. This connects the MDM training loss to any-order language models such as BERT (Devlin et al., 2019), which model the posterior marginals at all masked positions. We note that the training loss (1) allows other equivalent formulations, as developed in previous MDM works (Nie et al., 2025; Sahoo et al., 2024; Shi et al., 2024).

MDM inference. MDM inference starts from a length-L masked sequence $\mathbf{x}_1 = (\mathbf{m}, \ldots, \mathbf{m})$ and proceeds over a monotonically decreasing sequence of times $t_0 = 1 > \cdots > t_N = 0$. At each step t_ℓ, given partially masked sequence $\mathbf{x}_{t_\ell} \in (\mathcal{V} \cup \{\mathbf{m}\})^L$, we proceed by two steps to obtain $\mathbf{x}_{t_{\ell+1}}$:

(a) Choose a subset of masked tokens $\mathcal{S}$ to unmask, $\mathcal{S} \subseteq \{i \mid \mathbf{x}^i_{t_\ell} = \mathbf{m}\}$.

(b) For each $i \in \mathcal{S}$, unmask $\mathbf{x}^i_{t_\ell}$ to a clean token sampled from $f^i_\theta(\cdot \mid \mathbf{x}_{t_\ell}) \in \Delta(\mathcal{V})$.

Notably, step (a)—the choice of $\mathcal{S}$—is highly flexible and central to MDM's gains on downstream tasks; $\mathcal{S}$ may be chosen arbitrarily. Two predominant strategies are independently including each masked index $\mathbf{x}^i_{t_\ell} = \mathbf{m}$ with a fixed probability (Sahoo et al., 2024; Shi et al., 2024) or adaptively selecting indices, e.g., the most confident tokens or the leftmost tokens (Chang et al., 2022; Zheng et al., 2023; Kim et al., 2025; Nie et al., 2025).

2.2 Self-correction for MDM via per-token quality

An ideal generative model would detect low-quality tokens at inference time and correct them when appropriate. Prior work has explored self-correction in LLMs, including Kumar et al. (2024); Gou et al. (2023), and, recently, in MDMs. Despite varying instantiations, these methods for MDM can be organized into a common two-step procedure. At each inference step, given $\mathbf{x}_{t_\ell}$:

(a) Compute a per-token "quality score" for each $\mathbf{x}^i_{t_\ell} \neq \mathbf{m}$ and, according to it, select a subset of clean tokens $\mathcal{T}$ to remask.

(b) For each $i \in \mathcal{T}$, remask $\mathbf{x}^i_{t_\ell}$ and either replace it with a new clean or leave it masked.

At step (a), the set $\mathcal{T}$ is typically chosen with the lowest per-token quality, optionally with stochasticity. At step (b), the remasked $\mathbf{x}^i_{t_\ell}$ can be immediately resampled (using the pretrained model or an auxiliary module) or kept to engage further for future unmasking/remasking steps. Combined with the standard MDM unmasking step (Section 2.1), inference alternates between unmasking and (re)masking; these may be interleaved or staged, depending on the method.

Challenge. The key challenges are twofold: formulating a **principled definition of per-token quality** and **constructing a model** that estimates it. These can be formalized as follows: A per-token quality is a function $g_\star$ that takes any partially masked sequence $\mathbf{y}$ and returns position-wise scalars for non-masked positions $\mathbf{y}^i \neq \mathbf{m}$:

$$g_\star \colon (\mathcal{V} \cup \{\mathbf{m}\})^L \to [0,1]^L, \quad g^i_\star(\mathbf{y}) \approx \text{(Per-token quality)} = (\text{"how likely" } \mathbf{y}^i \text{ is given } \mathbf{y}).$$

The phrase "how likely" is a modeling choice; whatever the choice, it should align with the intended notion of per-token quality (our PRISM instantiation is given in Section 3). The goal is a computationally efficient algorithm for modeling $g_\star$ that, in the limit, provably learns the chosen definition of per-token quality. As we argue next, prior work typically lacks either a precise, testable target of per-token quality or entirely changes the MDM design itself, limiting its applicability. We therefore organize the prior work along whether it specifies a precise target for per-token quality on the state during inference $\mathbf{x}_{t_j}$ and how disruptive it is to the MDM design.

- Training-free approaches: DFM (Gat et al., 2024), ReMDM (Wang et al., 2025), P2-Self (Peng et al., 2025) are drop-in and can be applied to any pretrained MDM, however they lack a precise target: scores are random (DFM, ReMDM), evaluated on a different input than $\mathbf{x}_{t_\ell}$ (ReMDM-conf), or inferred from logits that were not trained to present the token quality (P2-Self).
- Methods that require rehauling MDM design: GIDD (von Rütte et al., 2025), Informed Corrector (Zhao et al., 2024), indeed articulate a valid target for self-correction but achieve it by changing the pretraining design or architecture, requiring full retraining.
- Proxy-$\hat{\mathbf{x}}$ approaches: Token Critic (Gou et al., 2023) and P2-Train (Peng et al., 2025) train an external scorer to approximate the marginal likelihoods of a sampled clean sequence $\hat{\mathbf{x}}$. For a given $\mathbf{x}_{t_\ell}$, it first samples a reconstructed clean sequence $\hat{\mathbf{x}}$ via f_θ in a single step, then calculates the quality score via the trained module. Since the pretrained MDM has learned the *per-token*

unmasking posterior, not the *joint posterior* across positions, $\hat{\mathbf{x}}$ can deviate a lot from a valid sample from the posterior. This method also requires an extra forward pass for scoring.

We provide a comprehensive explanation of the prior work and their notions of per-token quality in Appendix A. *In short*, prior work either lacks a precise target of per-token quality or alters the design of the MDM itself. *In contrast*, PRISM assigns an explicit per-token target to a given sequence $\mathbf{x}_{t_\ell}$ and estimates it in a single forward pass by fine-tuning the pretrained MDM, while still preserving the MDM's ability to compute per-token posteriors.

3 PRISM

In this section, we introduce Plug-in Remasking for Inference-time Self-correction of Masked diffusion models (**PRISM**). We substantiate its theoretical foundation in Section 3.1, present our empirical modeling design in Section 3.2, and explain how PRISM is used during inference in Section 3.3. Recall the definition of per-token quality from Section 2.2:

$$g_\star \colon (\mathcal{V} \cup \{\mathbf{m}\})^L \to [0,1]^L, \quad g^i_\star(\mathbf{y}) \approx \text{(Per-token quality)} = (\text{“how likely” } \mathbf{y}^i \text{ is given } \mathbf{y}).$$

A natural candidate for the per-token quality is the ground-truth likelihood of the token $\mathbf{y}^i$ when $\mathbf{y}^i$ is masked out of $\mathbf{y}$. We denote this masked sequence by $\mathbf{y} \oplus \mathbf{m}_i$, i.e., $(\mathbf{y} \oplus \mathbf{m}_i)^i = \mathbf{m}$ and $(\mathbf{y} \oplus \mathbf{m}_i)^j = \mathbf{y}^j$ for $j \neq i$. Our desired notion of *per-token quality* is the likelihood of a given token conditioned on the rest of the sequence. When $\mathbf{y}^i$ is consistent with the context $\mathbf{y}$, this likelihood should be high; otherwise, it should be low, marking the position as a candidate for remasking. Formally, $g^i_\star(\mathbf{y}) := p(\mathbf{x}^i = \mathbf{y}^i \,|\, \mathbf{y} \oplus \mathbf{m}_i)$. To clarify, $p(\mathbf{x}^i = \cdot \,|\, \cdot)$ is the same as the unmasking posterior used for MDM in Section 2.1.

Challenge: Unmasking vs. Remasking training. Our goal is to train (or obtain) a model g_ϕ that approximates $g_\star$. This is challenging because g_ϕ must take the sequence where the position i is already unmasked, yet it must approximate the posterior defined with the *masked* context $\mathbf{y} \oplus \mathbf{m}_i$. In contrast, an MDM f_θ predicts from the masked position itself;

$$g^i_\phi(\mathbf{y}) \approx p(\mathbf{x}^i = \mathbf{y}^i \,|\, \mathbf{y} \oplus \mathbf{m}_i) \in [0,1], \quad f^i_\theta(\cdot \,|\, \mathbf{z}) \approx p(\mathbf{x}^i = \cdot \,|\, \mathbf{z}) \in \Delta(\mathcal{V}),$$

Although $g^i_\phi(\mathbf{y})$ and the $f^i_\theta(\cdot \,|\, \mathbf{y} \oplus \mathbf{m}_i)$'s $\mathbf{y}^i$-th coordinate target the same quantity, they condition on different inputs. Consequently, g_ϕ cannot be simply trained or efficiently queried as in MDM.

A naive workaround is a distillation-based approach, supervising $g^i_\phi(\mathbf{y})$ with the teacher signal $f^i_\theta(\cdot \,|\, \mathbf{y} \oplus \mathbf{m}^i)$. This has two drawbacks: First, even in an infinite data and compute regime, we cannot achieve perfect $g_{\phi^\star}$ if the teacher model f_θ is imperfect. Second, it is data-inefficient: each training pair $(\mathbf{y}, i)$ requires a fresh evaluation of f_θ on $\mathbf{y} \oplus \mathbf{m}_i$. Consequently, one forward pass of f supervises only a single index, i.e., covering m clean token positions in a sequence requires m passes.

To address this input mismatch, Zhao et al. (2024) leverage a hollow-transformer $g_{\phi'}$, with the property $g^i_{\phi'}(\mathbf{y} \oplus \mathbf{m}_i) \approx p(\mathbf{x}^i = \mathbf{y}^i \,|\, \mathbf{y} \oplus \mathbf{m}_i)$, enabling the use of the same loss as MDM. However, this requires adopting a new architecture (with a tweak on the attention mechanism) that processes $\mathbf{y}$ while equivalently behaving as if position i were masked, which entails retraining and limiting plug-and-play applicability to pretrained MDMs. As we will show, PRISM does not possess these limitations: it is plug-and-play with any pretrained MDM f_θ (no architectural changes), and—even when f_θ is imperfect, it recovers the desired per-token quality g_ϕ in the infinite-data limit.

3.1 Theoretical foundation of PRISM

In this section, we establish the theoretical foundation of PRISM. We begin by recalling the premise behind the MDM training loss before describing how we adapt this premise to give rise to PRISM.

Marginalization perspective on MDM training. We now aim to understand why the MDM training loss $\mathcal{L}(\theta)$ has the unmasking posterior as its unique minimizer. First, the expectation over the joint distribution of $(\mathbf{x}, \mathbf{z})$ can be written by first sampling $\mathbf{z}$ and then $\mathbf{x}$ from the induced posterior. For fixed $\mathbf{z}$, the loss only depends on the posterior of $\mathbf{x}^i$ given $\mathbf{z}$, which is exactly the unmasking

posterior $p(\mathbf{x}^i = \cdot \mid \mathbf{z})$. Thus, we can rewrite:

$$\mathcal{L}(\theta) = \mathbb{E}_{\mathbf{z}} \left[\frac{1}{n} \sum_{i:\, \mathbf{z}^i = \mathbf{m}} \mathbb{E}_{v \sim p(\mathbf{x}^i = \cdot \mid \mathbf{z})} \left[-\log f_\theta^i(v \mid \mathbf{z}) \right] \right],$$

and then we observe that the blue-colored term is the cross-entropy between two distributions $f_\theta^i(\cdot \mid \mathbf{z})$ and $p(\mathbf{x}^i = \cdot \mid \mathbf{z})$, which is minimized at $f_{\theta^\star}^i(\cdot \mid \mathbf{z}) = p(\mathbf{x}^i = \cdot \mid \mathbf{z})$ for every $(\mathbf{z}, i)$. Put differently, the masking process of MDM defines the joint distribution over $(\mathbf{x}, \mathbf{z})$ and fixing $\mathbf{z}$ and *marginalizing* over $\mathbf{x}$ yields a cross-entropy between the network output and the desired unmasking posterior. We use this marginalization technique to build PRISM.

PRISM. Recall our goal for training g_ϕ to predict per-token quality, formally defined as $g_{\phi^\star}^i(\mathbf{y}) = p(\mathbf{x}^i = \mathbf{y}^i \mid \mathbf{y} \oplus \mathbf{m}_i)$. Given a pretrained MDM f_θ, PRISM proceeds as follows.

(a) Construct a pair $(\mathbf{x}, \mathbf{z})$ as in MDM training: sample $\mathbf{x} \sim p_{\text{data}}$ and then draw $\mathbf{z}$ by masking.

(b) Sample $\mathbf{y}$: choose a masked index i from $\mathbf{z}$ and replace it with a clean token $\mathbf{y}^i \sim f_\theta^i(\cdot \mid \mathbf{z})$.

Step (a) defines the same joint distribution over $(\mathbf{x}, \mathbf{z})$ as MDM. Step (b) yields a random variable $(\mathbf{y}, i)$ conditioned on $\mathbf{z}$. Although we do not designate a sampling rule for the index i, PRISM's guarantee is invariant to its rule, e.g., uniformly random, confidence-based. As a result, PRISM induces a joint distribution over $(\mathbf{x}, \mathbf{z}, (\mathbf{y}, i))$. Taking expectation over this joint distribution, we define the PRISM loss:

$$\mathcal{L}(\phi) := \mathbb{E}_{\mathbf{x}, \mathbf{z}, (\mathbf{y}, i)} \left[\mathrm{BCE}(\mathbf{1}[\mathbf{x}^i = \mathbf{y}^i], g_\phi^i(\mathbf{y})) \right], \tag{2}$$

where BCE denotes a Binary Cross-Entropy loss $\mathrm{BCE}(b, p) = -b \log p - (1-b) \log(1-p)$ for a binary label $b \in \{0, 1\}$. This loss is the crux of PRISM, whose unique minimizer is the true per-token quality.

Proposition 1 (PRISM). *The per-token quality uniquely minimizes the PRISM loss $\mathcal{L}(\phi)$ (2):*

$$g_{\phi^\star}^i(\mathbf{y}) = p(\mathbf{x}^i = \mathbf{y}^i \mid \mathbf{y} \oplus \mathbf{m}_i).$$

Before proving Proposition 1, we highlight two key advantages that PRISM offers.

Advantages. First, the minimizer $g_{\phi^\star}^i(\mathbf{y})$ *does not* depend on the specific choice of f_θ; f_θ affects the training objective only through the marginalized distribution of $(\mathbf{y}, i)$, not the minimizer. This contrasts with the aforementioned distillation-based approach, in which an imperfect f_θ will not give rise to a perfect g_ϕ. Second, PRISM naturally adapts to a pretrained MDM; a single model can share the same architecture and carry two heads, one for the unmasking posterior and one for per-token quality. We detail the modeling choices of PRISM in Section 3.2.

PRISM: Guarantee. Proposition 1 can be demonstrated within a few lines. The key mechanism is that the joint distribution $(\mathbf{x}, \mathbf{z}, (\mathbf{y}, i))$ exhibits a simple posterior given $(\mathbf{y}, i)$; $\mathbf{z}$ is deterministically $\mathbf{z} = \mathbf{y} \oplus \mathbf{m}_i$ and each $\mathbf{x}^i$ is drawn from the same unmasking posterior of MDM, i.e., $v \sim p(\mathbf{x}^i = \cdot \mid \mathbf{z})$. Therefore, fixing $(\mathbf{y}, i)$ and then *marginalizing* the binary label $\mathbf{1}[\mathbf{x}^i = \mathbf{y}^i]$ over $(\mathbf{x}, \mathbf{z})$ yields the simple byproduct,

$$q := \mathbb{E}_{v \sim p(\mathbf{x}^i = \cdot \mid \mathbf{y} \oplus \mathbf{m}_i)} \left[\mathbf{1}[v = \mathbf{y}^i] \right] = p(\mathbf{x}^i = \mathbf{y}^i \mid \mathbf{y} \oplus \mathbf{m}_i),$$

the per-token quality. We then expand the binary cross-entropy loss:

$$\mathcal{L}(\phi) = \mathbb{E}_{(\mathbf{y}, i)} \left[-q \log(g_\phi^i(\mathbf{y})) - (1-q) \log(1 - g_\phi^i(\mathbf{y})) \right],$$

which is exactly minimized at $g_{\phi^\star}^i(\mathbf{y}) = p(\mathbf{x}^i = \mathbf{y}^i \mid \mathbf{y} \oplus \mathbf{m}_i)$.

3.2 Empirical design choice of PRISM

In this section, we present our empirical modeling choices that go into the design of PRISM. We first explain the plug-and-play aspect of PRISM, in which we adapt a pretrained MDM and fine-tune it via the PRISM loss.

Plug-and-Play approach. Assume a pretrained MDM f_θ is given, which by design has learned the unmasking posterior. PRISM augments this model with a lightweight auxiliary adapter that reuses

the same backbone to produce a size-L tensor. In words, the unmasking posterior and per-token quality share a single backbone and are computed by separate heads (Figure 1, right).

For a given sequence, we respectively denote the two head outputs as f_θ and g_θ. Let $\mathbf{h}_\theta$ denote the final hidden state of the MDM backbone. The unmasking posterior f_θ is obtained by passing $\mathbf{h}_\theta$ through the unmasking head (which originally exists in pretrained MDM) and then applying the softmax. The per-token quality g_θ is obtained by passing $\mathbf{h}_\theta$ through the attached head and applying the sigmoid. Sigmoid is used to guarantee $g_\theta^i \in [0, 1]$, making it suitable for the binary cross-entropy loss. Thus, with a single θ and a lightweight adapter, we *jointly model* the unmasking posterior and then the per-token quality: for a partially masked sequence $\mathbf{w}$,

$$g_\theta^i(\mathbf{w}) \approx p(\mathbf{x}^i = \mathbf{w}^i \mid \mathbf{w} \oplus \mathbf{m}_i) \in [0, 1], \quad f_\theta^i(\cdot \mid \mathbf{w}) \approx p(\mathbf{x}^i = \cdot \mid \mathbf{w}) \in \Delta(\mathcal{V}).$$

At a high level, the PRISM recipe is simple: train g_θ using the PRISM loss defined in (2). Consequently, the parameters updated through g_θ are those of the attached head and the backbone, excluding the unmasking head. As an alternative to fine-tuning the backbone, one can add a LoRA adapter (Hu et al., 2022) into the backbone and train only the LoRA adapter parameters.

Practical interventions. We introduce two practical interventions that improve PRISM's efficiency. First, to preserve f_θ's ability to estimate the unmasking posterior while g_θ is learning toward the per-token quality, we add the MDM training loss (2) as a regularization term. For a regularization constant $\lambda > 0$, the total loss is therefore:

$$\mathcal{L}(\theta) = \mathbb{E}_{\mathbf{x},\mathbf{z},(\mathbf{y},i)} \left[\mathrm{BCE}(\, \mathbf{1}[\mathbf{x}^i = \mathbf{y}^i] \,,\, g_\theta^i(\mathbf{y}) \,) + \lambda \times \frac{1}{n} \sum_{j\,:\,\mathbf{z}^j = \mathbf{m}} -\log f_\theta^j(\mathbf{x}^j \mid \mathbf{z}) \right].$$

Second, in the sampling process for constructing PRISM training pairs $(\mathbf{x}, \mathbf{z}, (\mathbf{y}, i))$, we modify step (b). To obtain $\mathbf{y}$, rather than unmasking a single $\mathbf{y}$ from $\mathbf{z}$, we obtain multiple $(\mathbf{y}, i)$ pairs from one $\mathbf{z}$. This produces multiple training pairs $(\mathbf{x}, \mathbf{z}, (\mathbf{y}, i))$ with varying $(\mathbf{y}, i)$ values from a single forward pass of $f_\theta(\cdot \mid \mathbf{z})$, improving data efficiency. Since this procedure of sample construction minimizes g_θ (not f_θ), we apply stop-gradient to this f_θ. The pseudocode is given as Algorithm 1.

Practical advantages. PRISM formulation not only guarantees to minimize the ground truth per-token quality but also has two practical advantages. First, since f_θ is already pretrained, its hidden states encode useful representations, likely accelerating g_θ's training on the per-token quality. Second, the targeted per-token quality is a scalar per position, in contrast to the per-position distribution by the unmasking posterior, making it substantially easier to learn. Perhaps surprisingly, we observe that using much less compute compared to MDM pretraining already yields a meaningful learning of the per-token quality (which we detail these results in Section 4), reinforcing the perspective that PRISM is a cheap *fine-tuning* intervention.

3.3 Employing PRISM at inference

In this section, we explain how a model resulting from PRISM is used to perform remasking at inference. The high-level procedure follows Section 2.1; at each inference step, we perform remasking and unmasking simultaneously, using the learned unmasking posterior and per-token quality, respectively. Assume we are given a fine-tuned MDM θ (via PRISM loss) and a monotonically decreasing grid of times $t_0 = 1 > \cdots > t_N = 0$ for inference. We initialize $\mathbf{x}_{t_0} = (\mathbf{m}, \ldots, \mathbf{m})$. At each step t_ℓ, we obtain $\mathbf{x}_{t_{\ell+1}}$ by these following steps.

(a) Choose a subset of masked tokens $\mathcal{S}$ to unmask.

(b) Choose a subset of clean tokens $\mathcal{T}$ to remask with the lowest quality scores $g_\theta(\mathbf{x}_{t_\ell})$.

(c) For each $i \in \mathcal{T}$, remask $\mathbf{x}_{t_\ell}^i$ and for each $j \in \mathcal{S}$, unmask $\mathbf{x}_{t_\ell}^j$ to a clean token sampled from $f_\theta^j(\cdot \mid \mathbf{x}_{t_\ell}) \in \Delta(\mathcal{V})$.

We emphasize that $f_\theta(\cdot \mid \mathbf{x}_{t_\ell})$ and $g_\theta(\mathbf{x}_{t_\ell})$ are obtained from a *single* forward pass of the backbone. Consequently, our method does not add any computational overhead relative to methods without remasking (vanilla MDM inference) or training-free approaches, e.g, ReMDM. We provide the detailed choice of $|\mathcal{T}|$ and $|\mathcal{S}|$ in Appendix C.

4 EXPERIMENTS

In this section, we demonstrate that PRISM is a practically efficient and scalable approach across different domains and scales. In Section 4.1, we apply PRISM to a 30M-scale MDM for Sudoku. In Section 4.2, we apply PRISM to a 170M-scale MDM and evaluate its ability for natural language modeling, and in Section 4.3, we apply PRISM to LLaDA-8B-Instruct (Nie et al., 2025) and evaluate it on a Python coding task. We first outline the pipeline shared across all experiments.

Experiment pipeline. Starting from a pretrained MDM, we attach a lightweight adapter and fine-tune it with the PRISM loss. We compare against remasking baselines ReMDM (ReMDM-cap/ReMDM-loop) and ReMDM-conf (Wang et al., 2025), re-evaluated using the same backbone after PRISM fine-tuning. We summarize our experimental results as follows:

- **Efficiency**: The PRISM fine-tuned MDM has the ability to self-correct using much fewer samples compared to those used for pretraining.
- **Performance**: The PRISM-tuned MDMs outperform ReMDM and ReMDM-conf, indicating that PRISM learns our desired notion of per-token quality quite well.

4.1 MOTIVATING EXAMPLE: SUDOKU PUZZLE

In this section, we examine PRISM on Sudoku puzzles and show that the learned per-token quality operates as desired. Before moving on to the results, we use Sudoku as a *metaphorical testbed* to explain why ReMDM and ReMDM-conf can fail to capture the intended per-token quality.

Sudoku as a metaphorical testbed. ReMDM assigns token quality randomly, which can unnecessarily flag correct cells. ReMDM-conf sets the token quality to the likelihood at the time a cell was unmasked; this is computed on a previous Sudoku board state and can quickly become misaligned from the current board. In contrast, PRISM predicts per-token quality under the current board, thereby reliably identifying incorrect cells.

Setup. We construct a corpus of Sudoku puzzles (48k for training and 2k for evaluation), following the setup of (Ben-Hamu et al., 2025; Alp, 2024). Each board is tokenized as a length-89 sequence comprising 81 cell values and 8 [EOL] tokens that separate columns. Following the MDM pretraining recipe of Sahoo et al. (2024), we use a 28.6M Diffusion Transformer (DiT; Peebles & Xie (2023)) architecture as the backbone and pretrain it for 100k iterations (≈ 530 epochs). We then fine-tune it with the PRISM loss, additionally with a lightweight adapter (0.13M parameters).

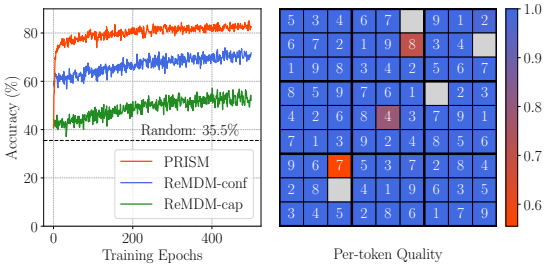

Figure 2: Left: PRISM achieves higher success accuracy on Sudoku puzzles than baselines. (red line) **Right:** PRISM detects the incorrect cells by assigning low per-token quality (red-colored cells).

Results. We evaluate the success rate for the same model across epochs for PRISM and baselines. As shown in Figure 2, PRISM achieves progressively higher success rates during fine-tuning (red line) and surpasses the baselines. This confirms that the learned per-token quality enables effective self-correction at inference. Notably, PRISM starts to outperform vanilla MDM inference (dashed line) and baselines within only a few epochs, far fewer than the pretraining epochs (≈ 530), corroborating our claim of sample efficiency.

Visualization of learned per-token quality. Figure 2 further visualizes PRISM's ability to detect incorrect tokens. On the shown board, three digits are misfilled: $(2, 6) = 8$, $(5, 5) = 4$, and $(7, 3) = 7$. The computed per-token quality assigns low scores to these cells (red), while other cells attain high scores, nearly 0.99 (blue), indicating that the fine-tuned model has almost perfectly learned the per-token quality.

4.2 UNCONDITIONAL TEXT GENERATION

Setup. We first take a DiT-based MDM with 170M parameters, using a checkpoint from (Sahoo et al., 2024). This MDM is pretrained on the OpenWebText corpus (Gokaslan & Cohen, 2019)

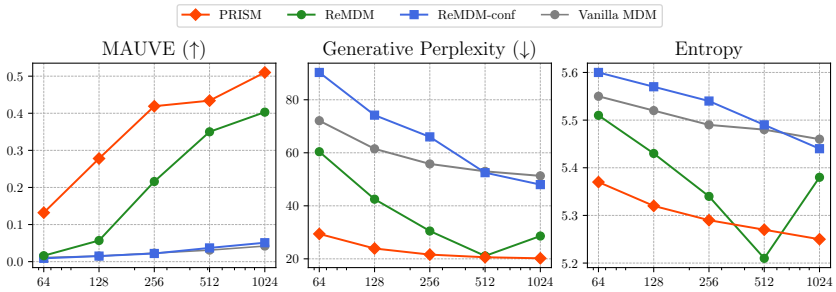

Figure 3: Unconditional text generation performance. Metrics are evaluated at NFE $\in \{64, 128, 256, 512, 1024\}$; PRISM (red) outperforms baselines (ReMDM: green, ReMDM-conf: blue, Vanilla MDM (without remasking): gray), particularly at lower sampling steps. Detailed numerical results are reported in Table 2.

with the GPT-2 tokenizer (Radford et al.), a maximum sequence length $L = 1024$, and a total of 262B tokens. We then attach a lightweight 7M parameter adapter, one attention layer applied to the final hidden state, and fine-tune it with the PRISM loss on a total 164M tokens from the same OpenWebText corpus. This fine-tuning is data-efficient, using $\mathbf{1600}\times$ fewer tokens than pretraining. During fine-tuning, we select the subset $\mathcal{S}$ based on confidence scores.

Results. We then assess the performance of PRISM and baselines on unconditional generation. Our primary metrics are Generative Perplexity and MAUVE (Pillutla et al., 2021), which respectively assess per-sequence fluency and distributional similarity between generated samples and the reference distribution. Moreover, given the prior observation (Zheng et al., 2024) that a sequence with too many redundant tokens can spuriously yield better, i.e., lower generative perplexity, we also report the entropy as a sanity metric. Following standard evaluation protocols (Pillutla et al., 2021; Wang et al., 2025), we generate 5000 samples to compute both metrics, using GPT-2-Large as the reference model. As shown in Figure 3, PRISM outperforms baselines on both metrics (red lines) while achieving a comparable entropy, especially in the regime with fewer sampling steps. See Appendix D.1 for additional details. We also provide a calibration analysis in Appendix D.5.

4.3 Applying PRISM to 8B MDM

Setup. We take LLaDA-8B-Instruct (Nie et al., 2025), an open-sourced 8B MDM. While freezing the backbone, we add an additional head and a LoRA adapter (Hu et al., 2022), for a total of $\approx$ 250M trainable parameters. We fine-tune it using the PRISM loss on the opc-sft-stage2 (Huang et al., 2024) (0.1M pairs) for 100 epochs, which takes $\approx$ 30 hours on 12 NVIDIA H100 GPUs.

	HumanEval			MBPP		
Sampling steps	256	512	1024	256	512	1024
PRISM	**28.0**	**39.0**	**42.7**	**21.8**	**29.1**	32.3
ReMDM	26.2	34.8	42.5	19.7	28.5	**32.9**
ReMDM-conf	25.6	34.1	36.6	20.1	29.0	32.5
MDM	25.6	34.1	36.6	18.2	27.8	31.9

Table 1: Zero-shot performance on coding tasks.

Results. We evaluate the resulting model in a zero-shot setting on MBPP (Austin et al., 2021b) and HumanEval (Chen, 2021), a challenging Python coding benchmarks. As shown in Table 1, PRISM outperforms ReMDM and ReMDM-conf across different numbers of sampling steps. We detail training and inference configurations in Appendix D.7, D.8.

Qualitative analysis. To assess the actual error-correction behavior of PRISM, we measure how often it corrects the syntax errors. Across 164 problems in HumanEval, PRISM corrects 19 out of 26 syntax errors. This provides fine-grained evidence that PRISM can identify and remediate errors in practice. For clarity, we report several failure and success instances in Appendix D.6.

5 Conclusion

We introduced PRISM, a plug-and-play fine-tuning framework that equips pretrained MDMs with self-correction ability, addressing prevalent limitations of prior work. PRISM operates under a theoretically grounded fine-tuning loss and shows efficiency across multiple domains and scales.

REFERENCES

Michael S. Albergo and Eric Vanden-Eijnden. Building normalizing flows with stochastic interpolants, 2022.

Michael S Albergo, Nicholas M Boffi, and Eric Vanden-Eijnden. Stochastic interpolants: A unifying framework for flows and diffusions. *arXiv preprint arXiv:2303.08797*, 2023.

Ali Alp. Sudoku. *https://github.com/alicommit-malp/sudoku/tree/main*, 2024.

Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne Van Den Berg. Structured denoising diffusion models in discrete state-spaces. *Advances in neural information processing systems*, 34:17981–17993, 2021a.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021b.

Heli Ben-Hamu, Itai Gat, Daniel Severo, Niklas Nolte, and Brian Karrer. Accelerated sampling from masked diffusion models via entropy bounded unmasking. *arXiv preprint arXiv:2505.24857*, 2025.

Andrew Campbell, Joe Benton, Valentin De Bortoli, Thomas Rainforth, George Deligiannidis, and Arnaud Doucet. A continuous time framework for discrete denoising models. *Advances in Neural Information Processing Systems*, 35:28266–28279, 2022.

Andrew Campbell, Jason Yim, Regina Barzilay, Tom Rainforth, and Tommi Jaakkola. Generative flows on discrete state-spaces: Enabling multimodal flows with applications to protein co-design. *arXiv preprint arXiv:2402.04997*, 2024.

Huiwen Chang, Han Zhang, Lu Jiang, Ce Liu, and William T Freeman. Maskgit: Masked generative image transformer. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11315–11325, 2022.

Mark Chen. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Google DeepMind. Gemini diffusion, 2025. URL `https://blog.google/technology/google-deepmind/gemini-diffusion/`.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pp. 4171–4186, 2019.

Itai Gat, Tal Remez, Neta Shaul, Felix Kreuk, Ricky TQ Chen, Gabriel Synnaeve, Yossi Adi, and Yaron Lipman. Discrete flow matching. *Advances in Neural Information Processing Systems*, 37: 133345–133385, 2024.

Aaron Gokaslan and Vanya Cohen. Openwebtext corpus. 2019.

Shansan Gong, Ruixiang Zhang, Huangjie Zheng, Jiatao Gu, Navdeep Jaitly, Lingpeng Kong, and Yizhe Zhang. Diffucoder: Understanding and improving masked diffusion models for code generation. *arXiv preprint arXiv:2506.20639*, 2025.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. Critic: Large language models can self-correct with tool-interactive critiquing. *arXiv preprint arXiv:2305.11738*, 2023.

Emiel Hoogeboom, Didrik Nielsen, Priyank Jaini, Patrick Forré, and Max Welling. Argmax flows and multinomial diffusion: Learning categorical distributions. *Advances in neural information processing systems*, 34:12454–12465, 2021.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.

Siming Huang, Tianhao Cheng, Jason Klein Liu, Jiaran Hao, Liuyihan Song, Yang Xu, J. Yang, J. H. Liu, Chenchen Zhang, Linzheng Chai, Ruifeng Yuan, Zhaoxiang Zhang, Jie Fu, Qian Liu, Ge Zhang, Zili Wang, Yuan Qi, Yinghui Xu, and Wei Chu. Opencoder: The open cookbook for top-tier code large language models. 2024. URL https://arxiv.org/pdf/2411.04905.

Wonjun Kang, Kevin Galim, Seunghyuk Oh, Minjae Lee, Yuchen Zeng, Shuibai Zhang, Coleman Hooper, Yuezhou Hu, Hyung Il Koo, Nam Ik Cho, et al. Parallelbench: Understanding the trade-offs of parallel decoding in diffusion llms. *arXiv preprint arXiv:2510.04767*, 2025.

Jaeyeon Kim, Kulin Shah, Vasilis Kontonis, Sham Kakade, and Sitan Chen. Train for the worst, plan for the best: Understanding token ordering in masked diffusions. *arXiv preprint arXiv:2502.06768*, 2025.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, et al. Training language models to self-correct via reinforcement learning. *arXiv preprint arXiv:2409.12917*, 2024.

Inception Labs, Samar Khanna, Siddhant Kharbanda, Shufan Li, Harshit Varma, Eric Wang, Sawyer Birnbaum, Ziyang Luo, Yanis Miraoui, Akash Palrecha, et al. Mercury: Ultra-fast language models based on diffusion. *arXiv preprint arXiv:2506.17298*, 2025.

José Lezama, Huiwen Chang, Lu Jiang, and Irfan Essa. Improved masked image generation with token-critic. In *European Conference on Computer Vision*, pp. 70–86. Springer, 2022a.

Jose Lezama, Tim Salimans, Lu Jiang, Huiwen Chang, Jonathan Ho, and Irfan Essa. Discrete predictor-corrector diffusion models for image synthesis. In *The Eleventh International Conference on Learning Representations*, 2022b.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling, 2022.

Anji Liu, Oliver Broadrick, Mathias Niepert, and Guy Van den Broeck. Discrete copula diffusion. *arXiv preprint arXiv:2410.01949*, 2024a.

Sulin Liu, Juno Nam, Andrew Campbell, Hannes Stärk, Yilun Xu, Tommi Jaakkola, and Rafael Gómez-Bombarelli. Think while you generate: Discrete diffusion with planned denoising. *arXiv preprint arXiv:2410.06264*, 2024b.

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

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

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. *arXiv preprint arXiv:2502.09992*, 2025.

Jingyang Ou, Shen Nie, Kaiwen Xue, Fengqi Zhu, Jiacheng Sun, Zhenguo Li, and Chongxuan Li. Your absorbing discrete diffusion secretly models the conditional distributions of clean data. *arXiv preprint arXiv:2406.03736*, 2024.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 4195–4205, 2023.

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

Krishna Pillutla, Swabha Swayamdipta, Rowan Zellers, John Thickstun, Sean Welleck, Yejin Choi, and Zaid Harchaoui. Mauve: Measuring the gap between neural text and human text using divergence frontiers. *Advances in Neural Information Processing Systems*, 34:4816–4828, 2021.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners.

Litu Rout, Constantine Caramanis, and Sanjay Shakkottai. Anchored diffusion language model. *arXiv preprint arXiv:2505.18456*, 2025.

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.

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.

Harshit Varma, Dheeraj Nagaraj, and Karthikeyan Shanmugam. Glauber generative model: Discrete diffusion models via binary classification. *arXiv preprint arXiv:2405.17035*, 2024.

Dimitri von Rütte, Janis Fluri, Yuhui Ding, Antonio Orvieto, Bernhard Schölkopf, and Thomas Hofmann. Generalized interpolating discrete diffusion. *arXiv preprint arXiv:2503.04482*, 2025.

Guanghan Wang, Yair Schiff, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Remasking discrete diffusion models with inference-time scaling. *arXiv preprint arXiv:2503.00307*, 2025.

Minkai Xu, Tomas Geffner, Karsten Kreis, Weili Nie, Yilun Xu, Jure Leskovec, Stefano Ermon, and Arash Vahdat. Energy-based diffusion language models for text generation. *arXiv preprint arXiv:2410.21357*, 2024.

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

Yixiu Zhao, Jiaxin Shi, Feng Chen, Shaul Druckmann, Lester Mackey, and Scott Linderman. Informed correctors for discrete diffusion models. *arXiv preprint arXiv:2407.21243*, 2024.

Kaiwen Zheng, Yongxin Chen, Hanzi Mao, Ming-Yu Liu, Jun Zhu, and Qinsheng Zhang. Masked diffusion models are secretly time-agnostic masked models and exploit inaccurate categorical sampling. *arXiv preprint arXiv:2409.02908*, 2024.

Lin Zheng, Jianbo Yuan, Lei Yu, and Lingpeng Kong. A reparameterized discrete diffusion model for text generation. *arXiv preprint arXiv:2302.05737*, 2023.

CONTENTS

A Related Works

A.1 Background and Related Literature

Masked Diffusion Models. Masked Diffusion Models (MDMs) were previously built on continuous-time Markov chains over discrete spaces (Hoogeboom et al., 2021). Subsequently, Austin et al. (2021a) introduced D3PM with several families of transition kernels, followed by SEDD (Lou et al., 2023). These discrete diffusions can also be understood through the lens of discrete flows (Campbell et al., 2024; Gat et al., 2024), which parallel flow matching and stochastic interpolants over continuous spaces (Albergo & Vanden-Eijnden, 2022; Albergo et al., 2023; Lipman et al., 2022). Among transition-kernel designs, the absorbing state (*masking*) kernel has been particularly popular due to its simple formulation and training objective. Subsequent work established a theoretical framework for these models (Zheng et al., 2023; Sahoo et al., 2024; Zheng et al., 2024). Follow-up works have scaled MDMs up to billions of parameters, showcasing their potential compared to autoregressive models (Nie et al., 2025; Ye et al., 2025; DeepMind, 2025; Labs et al., 2025; Gong et al., 2025).

Inference-time strategies for discrete diffusion models. A key feature of MDMs is their inference-time flexibility, highlighted by Zheng et al. (2024); Ou et al. (2024) and aligned with our any-order perspective in Section 2.1. Recall the MDM inference procedure from Section 2.1:

(a) Choose a subset of masked positions $\mathcal{S} \subseteq \{i \mid \mathbf{x}_{t_\ell}^i = \mathbf{m}\}$ to unmask.

(b) For each $i \in \mathcal{S}$, unmask by sampling a clean token from $f_\theta^i(\cdot \mid \mathbf{x}_{t_\ell}) \in \Delta(\mathcal{V})$ and setting $\mathbf{x}_{t_\ell}^i$ accordingly.

The choice of $\mathcal{S}$ is entirely flexible, and an appropriate selection can substantially improve downstream task performance. Since our work focuses on self-correction, i.e., re-masking, we only briefly survey prior work on unmasking strategies (Kim et al., 2025; Peng et al., 2025; Chang et al., 2022; Ben-Hamu et al., 2025; Rout et al., 2025). A complementary line of work studies inference-time planning strategies for discrete diffusion with uniform transition kernels (Liu et al., 2024b; Varma et al., 2024); these methods fall outside the scope of MDMs.

A.2 Details on prior work on remasking strategy (self-correction) for MDM

In this section, we detail our discussion in Section 2.2, focusing on their different notions of per-token qualities. We recall the notion of per-token quality stated in Section 2.2.

$$g_\star \colon (\mathcal{V} \cup \{\mathbf{m}\})^L \to [0,1]^L, \quad g_\star^i(\mathbf{y}) \approx \text{(Per-token quality)} = (\text{“how likely” } \mathbf{y}^i \text{ is given } \mathbf{y}).$$

- **DFM** (Gat et al., 2024) and **ReMDM** (Wang et al., 2025) randomly remask clean tokens in $\mathbf{y}$, which is equivalent to assigning i.i.d. per-token qualities $g_\star^i(\mathbf{y}) \sim \mathrm{Unif}[0,1]$.
- **P2-Self** (Peng et al., 2025) defines per-token quality using the pretrained MDM's score for the observed token under input $\mathbf{y}$, e.g., the probability $f_\theta^i(\mathbf{y}^i \mid \mathbf{y})$. Given that $\mathbf{y}^i \neq \mathbf{m}$, the model was never trained on this index in this state; it only encodes a meaningful quantity (unmasking posterior) when $\mathbf{y}^i = \mathbf{m}$. This notion of per-token quality is therefore not grounded.
- **ReMDM-conf** (Wang et al., 2025) takes the per-token quality to be the likelihood at the step the token was unmasked: if position i was unmasked at $t_k > t_\ell$ (i.e., an earlier step), set $g_\star^i := f_\theta^i(\mathbf{x}_{t_k}^i \mid \mathbf{x}_{t_k}) = f_\theta^i(\mathbf{x}_{t_\ell}^i \mid \mathbf{x}_{t_k})$. As noted, this ignores the current state $\mathbf{x}_{t_\ell}$ and can therefore be misaligned with the model's present beliefs.
- **Token Critic** (Lezama et al., 2022a;b) and **P2-train** (Peng et al., 2025) train an auxiliary scorer for the likelihood of a given prediction of clean sequence $\hat{\mathbf{x}}$. Given $\mathbf{x}_{t_\ell}$, they first form a full completion $\hat{\mathbf{x}}$ by unmasking each mask token at position i from $f_\theta^i(\cdot \mid \mathbf{x}_{t_\ell})$, then evaluate quality by passing this $\hat{\mathbf{x}}$ to the external module. A key limitation is that $\hat{\mathbf{x}}$ is drawn from per-token unmasking posteriors, not the ground-truth posterior across positions; thus, even with a perfect pretrained MDM, $\hat{\mathbf{s}}$ can deviate substantially from the true likelihood $\mathbf{x} \sim p_{\mathrm{data}}$, losing a theoretical guarantee.
- **GIDD** (von Rütte et al., 2025) introduces a family of transition kernels that linearly combine uniform and absorbing (masking) kernels. Models pretrained under this kernel encode posteriors at clean-token positions–however, this requires training under the new kernel and is not directly applicable to an off-the-shelf pretrained MDM, altogether losing the benefit of MDM over other types of kernels.

- **Informed corrector** (Zhao et al., 2024) employs a hollow-transformer $g_{\phi'}$ satisfying $g^i_{\phi'}(\mathbf{y}) = g^i_{\phi'}(\mathbf{y} \oplus \mathbf{m}_i) \approx p(\mathbf{x}^i = \mathbf{y}^i \mid \mathbf{y} \oplus \mathbf{m}_i)$, enabling the same training loss as MDM. This, however, mandates a modified architecture (attention tweak) that processes $\mathbf{y}$ as if position i were masked, necessitating retraining.

In summary, while these methods improve over vanilla MDM inference without remasking, they either substantially reshape the MDM pipeline or rely on imprecise notions and proxies for per-token quality.

B Training algorithms and guarantees of PRISM

B.1 PRISM fine-tuning algorithm

Algorithm 1 PRISM fine-tuning from a pretrained MDM

Require: Pretrained MDM backbone f_θ, per-token quality head g_θ, optimizer $\mathcal{O}$, true distribution p_{data}, regularization constant $\lambda > 0$, number of unmasking tokens $k \in \mathbb{N}$.
Sample $\mathbf{x} \sim p_{data}$, n, and $\mathbf{z}$ from the masking process.
Initialize the loss $\mathcal{L}(\theta) \leftarrow 0$
while $\mathcal{L}(\theta)$ converges
  `# Sample` $\mathbf{y}$ `from` $\mathbf{z}$ `with` $f_{\text{sg}(\theta)}$
  Select a subset $\mathcal{S} \subseteq \{i \mid \mathbf{z}^i = \mathbf{m}\}$ with $|\mathcal{S}| = k$.
  Obtain $\mathbf{y}$ by replacing $\mathbf{z}^i = \mathbf{m}$ to $v^i \sim f^i_{\text{sg}(\theta)}(\cdot \mid \mathbf{z})$ for each $i \in \mathcal{S}$.
  `# Calculate Loss`
  $\mathcal{L}(\theta) \leftarrow \mathcal{L}(\theta) + \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} \text{BCE}(\, \mathbf{1}[\mathbf{x}^i = \mathbf{y}^i]\,,\, g^i_\theta(\mathbf{y})\,)$ ▷ PRISM Loss
  $\mathcal{L}(\theta) \leftarrow \mathcal{L}(\theta) + \lambda \times \frac{1}{n} \sum_{j:\mathbf{z}^j = \mathbf{m}} -\log f^j_\theta(\mathbf{x}^j \mid \mathbf{z})$. ▷ Regularization (MDM loss)
  $\theta \leftarrow \mathcal{O}(\theta, \mathcal{L}(\theta))$
end while

B.2 Marginal Equivalence of the Masking Process

We formalize that the two masking procedures stated in Section 2.1 induce the same conditional distribution of $\mathbf{z}$ given $\mathbf{x}$, hence the same joint distributions $(\mathbf{x}, \mathbf{z})$. It suffices to show they generate the same distribution over masked index sets. Consider (b): draw $t \sim \text{Unif}[0, 1]$ and for each i, replace $\mathbf{x}^i \in \mathcal{V}$ with $\mathbf{m}$ with probability t. The probability of n tokens to be masked is:

$$\int_0^1 \binom{L}{n} t^n (1-t)^{L-n} dt = \binom{L}{n} \text{B}(n+1,\, L-n+1) = \frac{1}{L+1},$$

where B denotes the Beta function. This coincides with procedure (a), where n is uniformly sampled from $\{0, 1, \ldots, L\}$.

B.3 PRISM provably learns the per-token quality

In this section, we restate Proposition 1 and detail its proof. The proof essentially follows Section 3.1, but we provide the detailed calculations here.

Proposition 1 (restated). Let the PRISM loss be

$$\mathcal{L}(\phi) = \mathbb{E}_{\mathbf{x},\mathbf{z},(\mathbf{y},i)} \left[\text{BCE}\big(\mathbf{1}[\mathbf{x}^i = \mathbf{y}^i],\, g^i_\phi(\mathbf{y})\big)\right], \quad \text{BCE}(b, p) = -\,b \log p - (1-b)\log(1-p).$$

Then the unique minimizer satisfies, for every i and $\mathbf{y}$,

$$g^i_{\phi^\star}(\mathbf{y}) \;=\; p\big(\mathbf{x}^i = \mathbf{y}^i \,\big|\, \mathbf{y} \oplus \mathbf{m}_i\big)\,.$$

Proof. First, the expectation over the joint distribution $(\mathbf{x}, \mathbf{z}, (\mathbf{y}, i))$ can be written by first sampling $(\mathbf{y}, i)$ and then $(\mathbf{z}, \mathbf{x})$ from the induced posterior. We observe that this induced posterior admits a handy expression:

$$\mathbf{z} = \mathbf{y} \oplus \mathbf{m}_i, \quad v \sim p(\mathbf{x}^i = \cdot \mid \mathbf{z}).$$

Here $p(\mathbf{x}^i = \cdot \mid \cdot)$ is the unmasking posterior given by the MDM's masking process (Section 3.1), and to avoid notational confusion, we introduce a dummy variable v. Next, we expand the expectation term.

$$\begin{aligned}\mathcal{L}(\phi) =& \mathbb{E}_{(\mathbf{y},i)}\left[\mathbb{E}_{v\sim p(\mathbf{x}^i=\cdot\,|\,\mathbf{z}),z=\mathbf{y}\oplus\mathbf{m}_i}\left[\mathrm{BCE}\big(\mathbf{1}[v=\mathbf{y}^i],\, g^i_\phi(\mathbf{y})\big)\right]\right]\\ =& \mathbb{E}_{(\mathbf{y},i)}\left[-q\log(g^i_\phi(\mathbf{y})) - (1-q)\log(1-g^i_\phi(\mathbf{y}))\right],\end{aligned}$$

where

$$q \coloneqq \mathbb{E}_{v\sim p(\mathbf{x}^i=\cdot\,|\mathbf{y}\oplus\mathbf{m}_i)}\left[\mathbf{1}[v=\mathbf{y}^i]\right] = p(\mathbf{x}^i=\mathbf{y}^i\,|\,\mathbf{y}\oplus\mathbf{m}_i).$$

Finally, we conclude that $g^i_{\phi^\star}(\mathbf{y}) = q = p(\mathbf{x}^i = \mathbf{y}^i\,|\,\mathbf{y}\oplus\mathbf{m}_i)$ since $f_q(x) = -q\log x - (1-q)\log(1-x)$ is minimized at $x = q \in (0,1)$. □

B.4 Extension on PRISM's provable guarantee

In this section, we provide the pseudocode of the practically used PRISM fine-tuning pipeline and then state its theoretical guarantee. We note that the choice of index set $\mathcal{S}$ at line 5 of Algorithm 1 is flexible; it can either be random or confidence-based. To clarify, at line 6, although we use f_θ to obtain $\mathbf{y}$, we use the stop-gradient operation.

Note that the process of obtaining $\mathbf{y}$ and $\mathcal{S}$ can be interpreted as a process of obtaining multiple $(\mathbf{y}, i)$ pairs for each $i \in \mathcal{S}$. Under this, the expectation over $(\mathbf{x}, \mathbf{z}, (\mathbf{y}, \mathcal{S}))$ is equivalent to the expectation over $(\mathbf{x}, \mathbf{z}, (\mathbf{y}, i))$. Therefore, both formulations of the loss function are valid and equivalent.

$$\mathcal{L}(\theta) = \mathbb{E}_{\mathbf{x},\mathbf{z},(\mathbf{y},\mathcal{S})}\left[\frac{1}{|\mathcal{S}|}\sum_{i\in\mathcal{S}}\mathrm{BCE}(\,\mathbf{1}[\mathbf{x}^i=\mathbf{y}^i]\,,\, g^i_\theta(\mathbf{y})\,)\right] = \mathbb{E}_{\mathbf{x},\mathbf{z},(\mathbf{y},i)}\left[\mathrm{BCE}(\,\mathbf{1}[\mathbf{x}^i=\mathbf{y}^i]\,,\, g^i_\theta(\mathbf{y})\,)\right].$$

Now we provide the theoretical guarantee on our PRISM loss.

Proposition 2 (Provable guarantee of PRISM-extension)**.** *For a fixed $1 \le k \le L$, the PRISM loss*

$$\mathcal{L}(\phi) = \mathbb{E}_{\mathbf{x},\mathbf{z},(\mathbf{y},i)}\left[\mathrm{BCE}(\,\mathbf{1}[\mathbf{x}^i=\mathbf{y}^i]\,,\, g^i_\phi(\mathbf{y})\,)\right]$$

has the unique minimizer

$$g^i_{\phi^\star}(\mathbf{y}) = \mathbb{E}_{\mathcal{S}\sim\Pi(\cdot|\mathbf{y},i)}\left[p(\mathbf{x}^i=\mathbf{y}^i\mid\mathbf{y}\oplus\mathbf{m}_\mathcal{S})\right],$$

where the expectation is taken over the posterior distribution over $\mathcal{S}$ given $\mathbf{y}$ and i, induced by the joint distribution above. Here $\mathbf{y}\oplus\mathbf{m}_\mathcal{S}$ is a sequence obtained from $\mathbf{y}$ by replacing its i-th token with $\mathbf{m}$ for every $i \in \mathcal{S}$, aligning the notation in Section 3.1.

Proof. Before proving this, we clarify some notations. The notation $\mathbf{y}\oplus\mathbf{m}_\mathcal{S}$ extends the notation $\mathbf{y}\oplus\mathbf{m}_i$ from Section 3.1 into the case where we mask multiple tokens, more precisely, $\mathbf{y}\oplus\mathbf{m}_{\{i\}}$.

Also, note that the posterior distribution $\mathcal{S}\sim\Pi(\cdot\mid\mathbf{y},i)$ is naturally induced from the defined joint distribution. To clarify, this posterior inherits a dependence on the rule that we choose the index set $\mathcal{S}$ during the fine-tuning (and also f_θ). This also satisfies the following property: for a fixed $(\mathbf{y},i)$, sample $\mathcal{S}$ from this posterior distribution $\Pi(\cdot\mid\mathbf{y},i)$, and then the desired $\mathbf{x}$'s per-position posterior follows $v\sim p(x^i=\cdot\,|\,y\oplus\mathbf{m}_\mathcal{S})$. This nice property holds since the sampling process of $(\mathbf{x},\mathbf{z},(\mathbf{y},i))$ forms a Markov chain–$\mathbf{z}$ is conditioned on $\mathbf{x}$ and $(\mathbf{y},i)$ is conditioned on $\mathbf{x}$.

By using this property, we now complete the proof; the remaining parts are the same as Proposition 1.

$$\begin{aligned}\mathcal{L}(\phi) =& \mathbb{E}_{\mathbf{x},\mathbf{z},(\mathbf{y},i)}\left[\mathrm{BCE}(\,\mathbf{1}[\mathbf{x}^i=\mathbf{y}^i]\,,\, g^i_\phi(\mathbf{y})\,)\right]\\ =& \mathbb{E}_{y,i}\mathbb{E}_{\mathcal{S}\sim\Pi(\cdot|\mathbf{y},i)}\mathbb{E}_{v\sim p(\mathbf{x}^i=\cdot\,|\,\mathbf{z}),\mathbf{z}=\mathbf{y}\oplus\mathbf{m}_\mathcal{S}}\left[\mathrm{BCE}(\,\mathbf{1}[v=\mathbf{y}^i]\,,\, g^i_\theta(\mathbf{y})\,)\right]\\ =& \mathbb{E}_{y,i}\mathbb{E}_{\mathcal{S}\sim\Pi(\cdot|\mathbf{y},i)}\left[-q\log(g^i_\phi(\mathbf{y})) - (1-q)\log(1-g^i_\phi(\mathbf{y}))\right],\end{aligned}$$

where $q = p(\mathbf{x}^i=\mathbf{y}^i\mid\mathbf{y}\oplus\mathbf{m}_\mathcal{S})$. Therefore, the expectation over $\mathcal{S}\sim\Pi(\cdot\mid\mathbf{y},i)$ *absorbs* into q, finally derived to our desired form;

$$\mathcal{L}(\phi) = \mathbb{E}_{\mathbf{x},\mathbf{z},(\mathbf{y},i)}\left[\mathrm{BCE}(\,\mathbf{1}[\mathbf{x}^i=\mathbf{y}^i]\,,\, g^i_\phi(\mathbf{y})\,)\right] = \mathbb{E}_{y,i}\left[-q'\log(g^i_\phi(\mathbf{y})) - (1-q')\log(1-g^i_\phi(\mathbf{y}))\right],$$
$$q' = \mathbb{E}_{\mathcal{S}\sim\Pi(\cdot|\mathbf{y},i)}\left[p(\mathbf{x}^i=\mathbf{y}^i\mid\mathbf{y}\oplus\mathbf{m}_\mathcal{S})\right].$$

□

Discussion on a new notion of per-token quality. Proposition 2 guarantees that the minimizer of the extended PRISM loss satisfies

$$g^i_{\phi^\star}(\mathbf{y}) = \mathbb{E}_{\mathcal{S}\sim\Pi(\cdot|\mathbf{y},i)}\left[p(\mathbf{x}^i = \mathbf{y}^i \mid \mathbf{y} \oplus \mathbf{m}_\mathcal{S})\right].$$

Note that by construction, we always have $i \in \mathcal{S}$. This encompasses the $k = 1$ case in Section 3.1; When $k = 1$, the posterior distribution $\Pi(\cdot \mid \mathbf{y}, i)$ collapses to a point mass at $\mathcal{S} = \{i\}$, recovering Proposition 1.

However, we emphasize that this extended definition still captures our *intended* notion of per-token quality—namely, the (posterior) likelihood of $\mathbf{y}^i$ given the rest of the context, here defined under a certain distribution over $\mathbf{y} \oplus \mathbf{m}_\mathcal{S}$. (note that always $i \in \mathcal{S}$). Consequently, a model fine-tuned with the (extended) PRISM loss retains the capacity to identify low-quality tokens, while allowing $k > 1$ to average over multiple plausible masked-context views of $\mathbf{y}$.

f_θ's effect on PRISM. We show in Proposition 1 and Proposition 2 that, the design of (b), in which we use f_θ to obtain $\mathbf{y}$, does not directly influence our theoretical results. To clarify, when $k = 1$, the minimizer $g^i_{\phi^\star}(\mathbf{y}) = p\big(\mathbf{x}^i = \mathbf{y}^i \,\big|\, \mathbf{y} \oplus \mathbf{m}_i\big)$ remains fully independent from f_θ (or the rule for choosing i), and when $k > 1$, the only change is indirect: f_θ (and the rule for choosing index set $\mathcal{S}$ during PRISM fine-tuning) affects the posterior over masked sets $\Pi(\cdot \mid \mathbf{y}, i)$.

Practically, f_θ *shapes* the training distribution of samples $(\mathbf{y}, i)$ observed during PRISM fine-tuning, potentially aligning it with those encountered at inference. Since the PRISM loss supervises $\mathbf{y}^i$, which is sampled from f_θ, it is likely to be seen at the inference time (since we also use f_θ during inference to sample clean tokens). We hypothesize that this alignment improves PRISM fine-tuning, and we revisit this point with an ablation on OpenWebText (Appendix D.4).

C EMPLOYING PRISM AT INFERENCE

In this section, we provide details and pseudocode of inference algorithms that we use for our experiments. We first recall from Section 3.3 how PRISM fine-tuned model (f_θ, g_θ) is used at inference: At each step t_ℓ, we obtain $\mathbf{x}_{t_{\ell+1}}$ through the following steps.

(a) Choose a subset of masked tokens $\mathcal{S}$ to unmask.

(b) Choose a subset of clean tokens $\mathcal{T}$ to remask with the lowest quality scores $g_\theta(\mathbf{x}_{t_\ell})$.

(c) For each $i \in \mathcal{T}$, remask $\mathbf{x}^i_{t_\ell}$ and for each $j \in \mathcal{S}$, unmask $\mathbf{x}^j_{t_\ell}$ to a clean token sampled from $f^j_\theta(\cdot \mid \mathbf{x}_{t_\ell}) \in \Delta(\mathcal{V})$.

We now provide the pseudocode for two different cases of PRISM implementation, depending on how we choose the sizes of the unmasking and remasking sets $|\mathcal{S}|$ and $|\mathcal{T}|$. We begin with the case where we stochastically assign $|\mathcal{S}|$ and $|\mathcal{T}|$ from binomial distributions (Algorithm 2), corresponding to our experiments on OpenWebText (Section 4.2).

Algorithm 2 PRISM inference with assignments from binomial distributions

Require: Fine-tuned MDM with unmasking posterior model f_θ, per-token quality head g_θ, discretized time steps $\{t_\ell\}_{\ell=0}^N$, sampling steps N, sequence length L, remasking token ratio $\eta > 0$.
Initialize $\mathbf{x}_1 = \mathbf{x}_{t_1} \leftarrow (\mathbf{m}, \ldots, \mathbf{m})$
for $\ell = 0$ to $N-1$
  $K \sim \mathrm{Binom}(|\{i \mid \mathbf{x}^i_{t_\ell} \neq \mathbf{m}\}|, \eta)$ and define $\mathcal{M} \leftarrow \{i \mid \mathbf{x}^i_{t_\ell} = \mathbf{m}\}$
  if $K \leq |\mathcal{M}| < L - K$ and $l \geq l_{on}$:
    $|\mathcal{T}| \leftarrow K$, $|\mathcal{S}| \leftarrow \lceil \frac{L}{N_s} \rceil + K$
  else:
    $|\mathcal{T}| \leftarrow 0$, $|\mathcal{S}| \leftarrow \lceil \frac{L}{N_s} \rceil$
  Sample $\mathcal{S} \subset \mathcal{M}$ according to the unmasking rule
  Sample $\mathcal{T} \subset \{i \mid \mathbf{x}^i_{t_\ell} \neq \mathbf{m}\}$ with low-$|\mathcal{T}|$ indices of $g_\theta(x_{t_\ell})$
  Unmask $\mathbf{x}^i_{t_\ell}$ to $v^i \sim f^i_\theta(\cdot \mid \mathbf{x}_{t_\ell})$ for each $i \in \mathcal{S}$
  Remask $\mathbf{x}^j_{t_\ell}$ for each $j \in \mathcal{T}$
  $\mathbf{x}_{t_{\ell+1}} \leftarrow \mathbf{x}_{t_\ell}$
return $\mathbf{x}_{t_N} = \mathbf{x}_0$

Now, we state the algorithm where we pre-allocate $|\mathcal{S}|$ and $|\mathcal{T}|$ before inference (Algorithm 3) with fixed K, corresponding to our experiments on Sudoku 4.1 and LLaDA 4.3.

Algorithm 3 PRISM inference with fixed K

Require: Fine-tuned MDM with unmasking posterior model f_θ, per-token quality head g_θ, sampling steps N, sequence length L, discretized time steps $\{t_\ell\}_{\ell=0}^N$, number of remasking tokens K
Initialize $\mathbf{x}_1 = \mathbf{x}_{t_1} \leftarrow (\mathbf{m}, \ldots, \mathbf{m})$
for $l = 0$ to $N-1$
  Define $\mathcal{M} \leftarrow \{i \mid \mathbf{x}_{t_l} = \mathbf{m}\}$
  if $K \leq |\mathcal{M}| < L - K$ and $l \geq l_{on}$:
    $|\mathcal{T}| = K$, $|\mathcal{S}| = \lceil \frac{L}{N} \rceil + K$
  else:
    $|\mathcal{T}| = 0$, $|\mathcal{S}| = \lceil \frac{L}{N} \rceil$
  Sample $\mathcal{S} \subset \mathcal{M}$ according to the unmasking rule
  Sample $\mathcal{T} \subset \{i \mid \mathbf{x}^i_{t_\ell} \neq \mathbf{m}\}$ with low-$|\mathcal{T}|$ indices of $g_\theta(x_{t_\ell})$
  Unmask $\mathbf{x}^i_{t_\ell}$ to $v^i \sim f^i_\theta(\cdot \mid \mathbf{x}_{t_\ell})$ for each $i \in \mathcal{S}$
  Remask $\mathbf{x}^j_{t_\ell}$ for each $j \in \mathcal{T}$
  $\mathbf{x}_{t_{\ell+1}} \leftarrow \mathbf{x}_{t_\ell}$
return $\mathbf{x}_{t_N} = \mathbf{x}_0$

D EXPERIMENT DETAILS AND ABLATION STUDIES

This section provides additional experimental details.

D.1 OMITTED RESULTS

Here, we present the comprehensive results on OpenWebText in terms of MAUVE score, Generative Perplexity (Gen PPL), and Entropy. The graph in Figure 3 is derived from Table 2. We select the subset $\mathcal{S}$ based on confidence scores during fine-tuning.

Table 2: Evaluation result for PRISM/baselines across N. The best MAUVE scores are **bolded**. Baseline results marked with † are taken from Wang et al. (2025).

Method	MAUVE (↑)					Gen PPL. (↓)					Entropy (↑)				
	64	*128*	*256*	*512*	*1024*	*64*	*128*	*256*	*512*	*1024*	*64*	*128*	*256*	*512*	*1024*
MDLM†	0.011	0.015	0.023	0.031	0.042	72.1	61.5	55.8	53.0	51.3	5.55	5.52	5.49	5.48	5.46
ReMDM-conf	0.009	0.015	0.022	0.037	0.051	90.3	74.2	66.0	52.5	48.0	5.60	5.57	5.54	5.49	5.44
ReMDM†	0.016	0.057	0.216	0.350	0.403	60.4	42.5	30.5	21.1	28.6	5.51	5.43	5.34	5.21	5.38
PRISM (ours)	**0.132**	**0.278**	**0.419**	**0.434**	**0.510**	29.4	23.9	21.6	20.6	20.2	5.37	5.32	5.29	5.27	5.25

D.2 DETAILS ON THE REMASKING STRATEGY FOR OPENWEBTEXT

This section provides a detailed elaboration of the sampling strategies employed for the unconditional text generation in Section 4.2. The main motivation for these interventions is to preserve sentence diversity, which is crucial for evaluated metrics.

- **Staged Remasking**: We restrict the self-correction mechanism to be activated only after a specific step ℓ_{on}. This restriction is motivated by the observation that low per-token quality scores in the initial sampling phase ($t > t_{\ell_{on}}$) are likely due to a lack of suffix context rather than token-level inaccuracies. Activating the correction mechanism only after a sufficient context prevents spurious modifications to an incomplete sequence.
- η **Inverse Scheduling**: Fixing η was found to degrade the diversity as the N increases. To mitigate this, an inverse scheduling rule was adopted, dynamically adjusting η based on N.

To avoid complicated hyperparameter designs, we set these hyperparameters to be a function of N. The remasking ratio and the remasking activation step are set to $\eta = \frac{2.56}{N}$ and $l_{on} = \lceil 0.5N \rceil$, respectively.

D.3 ABLATION ON FINE-TUNING HYPERPARAMETERS-(k, n_y)

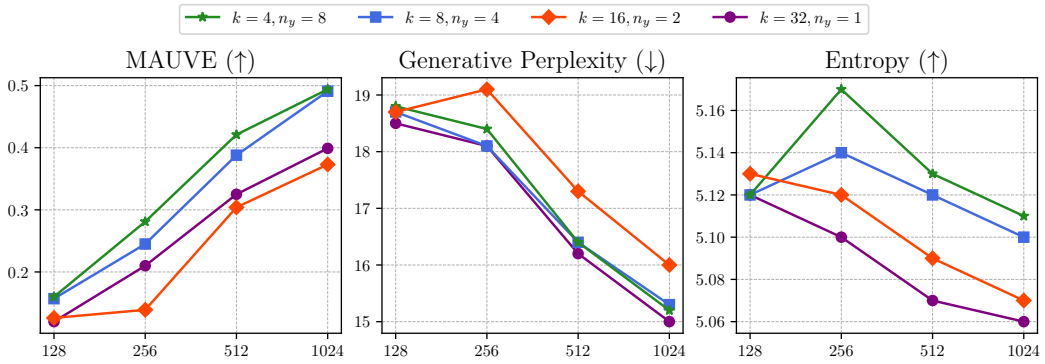

Figure 4: Ablation study on fine-tuning hyperparameters k and n_y while holding their product constant ($k \times n_y = 32$). Evaluations are conducted in $(128, 256, 512, 1024)$ sampling steps.

In addition to k (the number of tokens updated simultaneously to sample $\mathbf{y}$ from $\mathbf{z}$), we introduce a second hyperparameter, $n_y \in \mathbb{N}$, to improve data efficiency during the adapter's fine-tuning. As discussed in Section 3.2, we can generate multiple distinct target sequences $\mathbf{y}$ from a single source

sequence $\mathbf{z}$ by using different unmasking index sets $\mathcal{S}$. Therefore, n_y denotes the number of unique unmasking sets applied per $\mathbf{z}$ in a single forward pass.

We conducted an ablation study to analyze how the interplay between k and n_y affects the generation performance. We tested variants of $(k, n_y) \in \{(4, 8), (8, 4), (16, 2), (32, 1)\}$, thus remaining the total number of token updates per batch constant ($k \times n_y = 32$) for a fair comparison. All other configurations follow Table 4, except that we set nucleus $p = 0.9$ to sample $\mathbf{y}$ from $\mathbf{z}$, and select $\mathcal{S}$ randomly during PRISM fine-tuning.

Our primary finding is that training the adapter with a smaller number of updating tokens (low k) results in a model that performs better at inference time, as measured by the MAUVE score. This can be attributed to a train-test distribution mismatch induced by large k. Specifically, updating many tokens simultaneously is likely to introduce joint dependency errors, increasing the chance of sampling a *misleading* sequence $\mathbf{y}$. This forces the adapter to train on a corrupted data distribution that is rarely encountered during inference time. Consequently, this mismatch leads to a poorly calibrated quality estimator, degrading the model's ability to perform effective self-correction.

D.4 Ablation on Fine-tuning Hyperparameters-Nucleus sampling

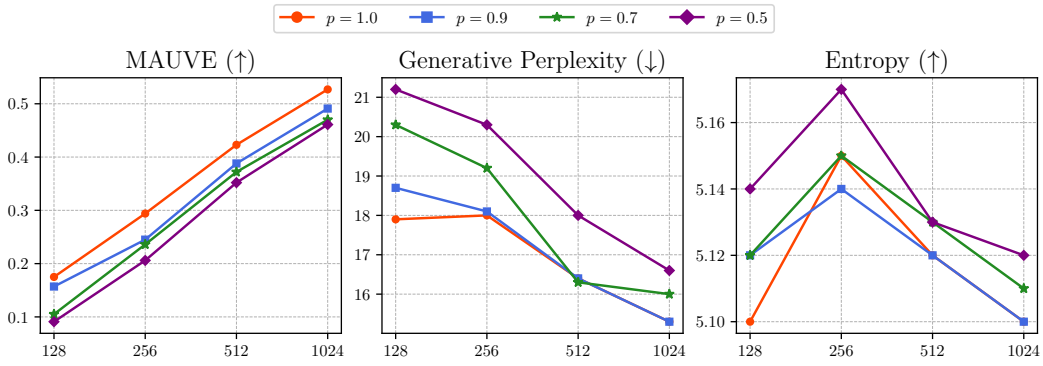

Figure 5: Ablation study on nucleus sampling p used to generate target sequences during adapter fine-tuning. Evaluations are conducted in $(128, 256, 512, 1024)$ sampling steps.

We further investigate how the sampling distribution used to generate target sequences $\mathbf{y}$ affects the adapter's fine-tuning process. Specifically, we analyze the impact of nucleus sampling probability p that we use to sample a token $\mathbf{y}^i$ from $f_\theta^i(\cdot \mid \mathbf{z})$, while selecting $\mathcal{S}$ randomly. Our hypothesis is that a less restrictive sampling distribution (large p) is more beneficial for training the quality estimator than a shrunk one (small p).

Intuitively, a large p induces a relatively diverse sample $\mathbf{y}$. These sequences include not only high-probability tokens but also rare yet plausible alternatives. Training on these plausible-but-imperfect examples reduces exposure bias by forcing the model to calibrate its confidence across a wider range of outcomes, rather than only learning from its own single best predictions. This process cultivates a more robust quality estimator capable of discerning subtle inaccuracies. While this approach may introduce gradients with larger variance, it provides superior coverage of the complex distributions encountered during inference. This ultimately leads to a more effective self-correction mechanism and improved text quality, as validated by the results in Fig. 5.

D.5 Quantitative analysis of PRISM: Calibration study

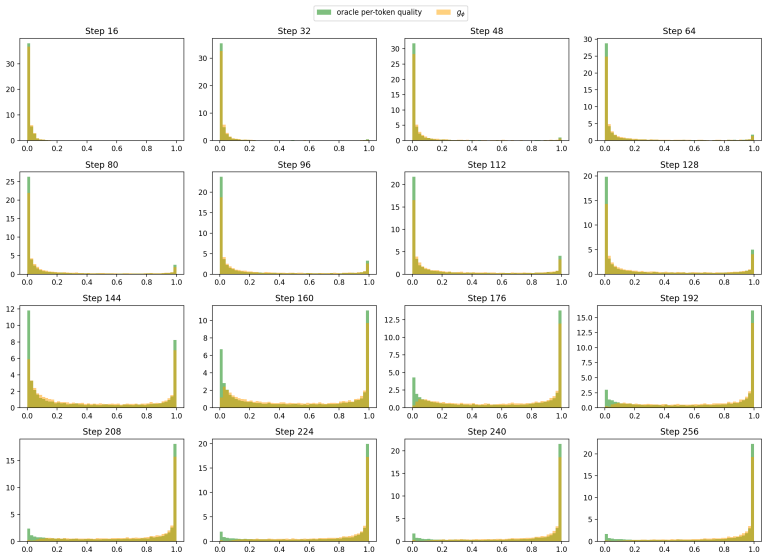

Figure 6: Calibration of PRISM per-token quality on OpenWebText during generation without remasking with 256 steps. For each timestep, tokens are binned by predicted quality $g_\theta^i(\mathbf{x}_t)$, and for each bin we plot the empirical probability that token i matches the ground truth, estimated via $f_\theta(\cdot \mid \mathbf{x}_t \oplus \mathbf{m}_i)$, against the bin mean.

We measure how well the learned per-token quality scores are calibrated to the targeted likelihood. Using the fine-tuned model from Section 4.2, we collect per-token quality scores $g_\theta^i(\mathbf{x}_t)$ for all inference-time sequences $\mathbf{x}_t$s and their clean positions i. Then we also compute the corresponding targeted likelihood queried from f_θ, namely $f_\theta^i(\mathbf{x}_t^i \mid \mathbf{x}_t \oplus \mathbf{m}_i)$. Surprisingly, the resulting calibration curves (Figure 6) closely track the queried likelihoods, indicating that $g_\theta^i(\mathbf{x}_t)$ closely calibrates the target posterior.

D.6 Quantitative analysis of PRISM: Error correction study

Error-correction analysis. To assess the actual error-correction behavior of PRISM, we take the model in Section 4.3 and measure how often it corrects the syntax errors. Across 164 problems in HumanEval, PRISM corrects 19 out of 26 syntax errors. This provides fine-grained evidence that PRISM can identify and remediate errors in practice. The code snippets are given in Table 3.

- Syntax error correction (task id 56): The baseline model produces a syntactically invalid program due to an incomplete `elif` statement, which results in a `SyntaxError`. PRISM successfully detects this low-quality token and replaces the invalid branch with a valid `else` clause, yielding a syntactically correct and functionally valid implementation.
- Logical error correction (task id 5): The baseline implementation incorrectly appends the delimiter after every element, including the last one. PRISM revises the code to conditionally insert the delimiter only between elements, correcting the logical structure of the program while preserving the intended functionality.
- (Failure case) overcorrection (task id 25): Although the baseline solution is correct, PRISM rewrites the code into a more complex form that relies on repeated `min` and `max` operations. This transformation introduces an error when the input list becomes empty, illustrating a failure mode where PRISM overcorrects a correct but non-canonical solution.
- (Failure case) global reasoning error (task id 17): Both the baseline model and PRISM fail on this task. The core issue lies in the iteration structure: the program should iterate over multi-character musical symbols rather than individual characters. This type of global structural error is not captured by PRISM's token-level quality estimation, highlighting a limitation of local self-correction.

Syntax error correction (task 56)

Baseline

```
def correct_bracketing(brackets):
    stack = []
    for bracket in brackets:
        if bracket == "<":
            stack.append(bracket)
        elif:   # SyntaxError
            if not stack:
                return False
            stack.pop()
    return not stack
```

PRISM

```
def correct_bracketing(brackets):
    stack = []
    for bracket in brackets:
        if bracket == "<":
            stack.append(bracket)
        else:
            if len(stack) == 0:
                return False
            stack.pop()
    return len(stack) == 0
```

Logical error correction (task 5)

Baseline

```
def intersperse(numbers,
↪  delimeter):
    result = []
    for num in numbers:
        result.append(num)
        result.append(delimeter)
    return result
```

PRISM

```
def intersperse(numbers,
↪  delimeter):
    result = []
    for num in numbers:
        if not result:
            result.append(num)
        else:
            result.append(delimete⌋
            ↪  r)
            result.append(num)
    return result
```

Failure case (task 25)

Baseline

```
def strange_sort_list(nums):
    if not nums:
        return []
    nums.sort()
    result = []
    while nums:
        result.append(nums.pop(0))
        if nums:
            result.append(nums.pop⌋
            ↪  (-1))
    return result
```

PRISM

```
def strange_sort_list(nums):
    result = []
    while nums:
        min_val = min(nums)
        result.append(min_val)
        nums.remove(min_val)
        max_val = max(nums)
        result.append(max_val)
        nums.remove(max_val)
    return result
```

Global reasoning failure (task 17)

Baseline

```
def parse_music(s):
    out = []
    for note in s:
        if note == 'o':
            out.append(4)
        elif note == 'o|':
            out.append(2)
        elif note == '.|':
            out.append(1)
    return out
```

PRISM

```
def parse_music(s):
    out = []
    for note in s:
        if note == 'o':
            out.append(4)
        elif note == 'o|':
            out.append(2)
        elif note == '.|':
            out.append(1)
    return out
```

Table 3: Qualitative comparison of self-correction behavior.

D.7 Experiment Configurations

In Table 4, we list the hyperparameter configurations for Sudoku, OWT, and LLaDA experiments.

D.8 Additional experimental details on LLaDA

Although Table 4 lists the basic configurations for our LLaDA experiments, we provide comprehensive details below.

Attaching the adapter. As discussed in Section 4.3, we freeze the LLaDA-8B-Instruct backbone and add (i) an auxiliary head to model per-token quality and (ii) a LoRA adapter on the qkv attention projections (rank 256, dropout ratio 0.1). This yields roughly 250M trainable parameters in total.

Table 4: Hyperparameter configurations for Sudoku, OWT, and LLaDA.

Hyperparameter	Sudoku	OWT	LLaDA
Base Model			
Architecture	DiT	DiT	(non-causual) Transfomer
Parameters	28.6M	169M	8B
Tokenizer	N/A	GPT-2	LLaDA-8B-Instruct
Sequence Length	81	1024	4096
Fine-tuning			
Adapter Architecture	linear	attention+linear	projection+linear
Adapter Parameters	133K	7.9M	240M
Optimizer	AdamW	AdamW	AdamW
Learning Rate	3.0×10^{-4}	1.5×10^{-4}	1.0×10^{-4}
Weight Decay	0.0	0.0	0.1
Gloabl Batch Size	256	256	144
k	4	32	8
n_y	1	1	1
Nucleus p	1.0	1.0	0.0
λ	5.0	0.5	0.5
Sampling			
unmasking strategy	random	random	semi-autoregressive
n_{blocks}	1	1	32
Nucleus p	1.0	0.9	0.0
Sampling Precision	float64	float64	float32
K	4	N/A	–
l_{on}	0	$\lceil 0.5N \rceil$	–
η	N/A	$\frac{2.56}{N}$	N/A

Dataset construction. For PRISM fine-tuning, we adopt the opc-sft-stage-2 (Huang et al., 2024), comprising ≈ 0.1M challenging Python coding problems. Each example includes instructions and solutions. Thus, in the masking process, we do *not* mask instruction tokens.

Training configuration. We use AdamW (Loshchilov & Hutter, 2017) (learning rate 1.0×10^{-4}, weight decay 0.01, warmup ratio 0.05) and use a cosine schedule with 5 cycles. Training for 100 epochs takes approximately 30 hours on 12 H100 GPUs.

Inference details. For unmasking, we follow Nie et al. (2025) and employ semi-autoregressive inference: the sequence is partitioned into multiple blocks, and decoding proceeds from the leftmost block. We set the nucleus parameter to $p = 0.0$. These two choices are known to be important for competitive coding performance.

For re-masking, across all baselines, we re-mask exactly K_{block} tokens per block and schedule these re-masking operations at intermediate stages of block-wise inference (we observe degraded performance if re-masking is concentrated only at the beginning or end of a block). For a fair comparison, we report the best performance for PRISM and baselines (ReMDM, ReMDM-conf) over a sweep of K_{block} at a fixed number of inference steps N: specifically, $K_{\text{block}} \in \{4, 6\}$ for $N = 256$, (this is because the number of unmasking steps equals 8 for $N = 256$) and $K_{\text{block}} \in \{8, 10, 12, 14, 16\}$ for $N = 512$ and $N = 1024$.