Learn from Your Mistakes: Self-Correcting Masked Diffusion Models

Yair Schiff [1,*] **Omer Belhasin** [2,*] **Roy Uziel** [2] **Guanghan Wang** [1] **Marianne Arriola** [1]

Gilad Turok [1] **Ran Zilberstein** [2] **Michael Elad** [2,†] **Volodymyr Kuleshov** [1,†]

[1] Cornell [2] NVIDIA
[*] Equal contribution [†] Equal senior authorship

Abstract

Masked diffusion models (MDMs) have emerged as a promising alternative to autoregressive models, enabling parallel token generation while achieving competitive performance. Despite these advantages, MDMs face a fundamental limitation: once tokens are unmasked, they remain fixed, leading to error accumulation and ultimately degrading sample quality. We address this by proposing a framework that trains a model to perform both unmasking and correction. By reusing outputs from the MDM denoising network as inputs for corrector training, we train a model to recover from potential mistakes. During generation we apply additional corrective refinement steps between unmasking ones in order to change decoded tokens and improve outputs. We name our training and sampling method *Pro*gressive *Se*lf-*Co*rrection (*ProSeCo*) for its unique ability to iteratively refine an entire sequence, including already generated tokens. We conduct extensive experimental validation across multiple conditional and unconditional tasks, demonstrating that *ProSeCo* yields better quality-efficiency trade-offs (up to ∼4x faster sampling) and enables inference-time compute scaling to further increase sample quality beyond standard MDMs (up to ∼1.2x improvement on benchmarks). We provide the code[1], model weights, and blog post on the project page: https://proseco-discrete-diffusion.github.io.

1 Introduction

Masked diffusion models (MDMs) [33, 37, 46, 51] have emerged as a powerful paradigm for discrete data generation and offer a compelling alternative to autoregressive (AR) models. Treating generation as a denoising process that gradually unmasks tokens in parallel, MDMs can achieve efficiency gains and maintain quality across various domains, and have even demonstrated competitive performance from the 8B scale [35] up to 100B parameters [7].

However, a fundamental limitation persists for MDMs: once a token is unmasked, it remains fixed for the duration of the generation process. Consequently, errors made during parallel decoding inevitably accumulate, leading to distributional drift and degraded sample quality. While recent work has begun to explore error correction for MDMs [23, 25, 30, 32, 57, 67], efficiently identifying which tokens require modification and altering them remains a significant challenge.

[1] Code: https://github.com/kuleshov-group/proseco

Correspondence to: Yair Schiff <yairschiff@cs.cornell.edu> and Omer Belhasin <obelhasin@nvidia.com>
Preprint under review.

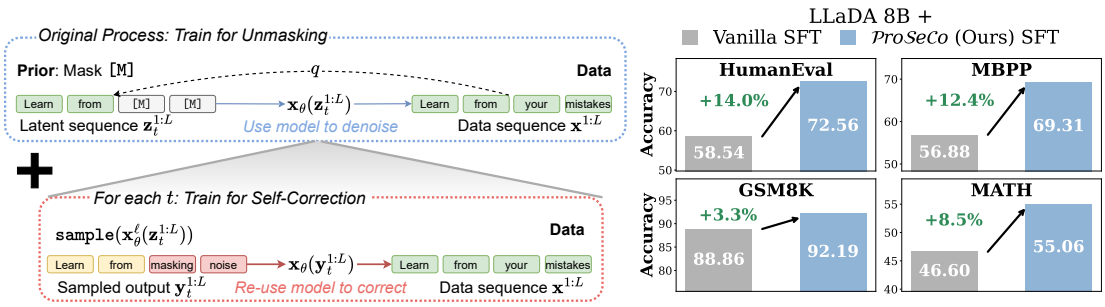

Figure 1: (*Left*) Training *ProSeCo*: The original process trains for generation via unmasking. For every timestep in the masking process, we also train for self-correction: undoing corruptions that arise from sampling from unmasking predictions. *(Right)* Using our method to supervised fine-tune (SFT) the 8B parameter LLaDA model [35] significantly outperforms vanilla masked diffusion SFT.

In this work, we address this limitation by proposing a principled framework that equips MDMs with the inherent ability to both decode and correct. Our key insight is to treat model-generated outputs as corrupted versions of the true data, where errors represent a form of noise that can be undone through a nested corrector loop. Our method therefore trains a model to recover the clean signal from its own potentially mistaken outputs, enabling it to learn from and correct its own failure modes. We implement this training via a simple additional cross-entropy loss term added to standard MDM objectives. During inference, we leverage this capability by interleaving corrective steps in between standard unmasking steps, allowing the model to dynamically refine and "self-correct" outputs. The training and inference modifications we make to existing MDM algorithms are minimal, yet lead to marked gains. We name our approach *Pro*gressive *Se*lf-*Co*rrection (*ProSeCo*), since our method has the ability to iteratively refine all positions in a sequence, including ones already unmasked.

On math and coding benchmarks, our method significantly improves standard fine-tuning of large MDMs, enabling up to ∼4x faster generation without quality degradation, and up to ∼1.2x increase in benchmark accuracy. For guided generation, we demonstrate that recovering from mistaken tokens improves the Pareto frontier of sample quality and property maximization. Finally, we also demonstrate that for unconditional generation, our method improves over MDMs and other proposed correctors in terms of generating fluent text without collapsing the diversity of generated outputs. Across all experiments we demonstrate that *ProSeCo* better trades-off quality and speed, and can also benefit from inference-time scaling to further increase sample quality.

In summary, our contributions are as follows: 1) We present a framework that jointly trains for decoding masked tokens and correcting mistakes. 2) We provide easy-to-implement training and sampling algorithms that entail only minor additions to standard MDMs. 3) We conduct comprehensive experiments demonstrating that *ProSeCo* outperforms quality-efficiency trade-offs of baseline methods and enables inference-time compute scaling to further improve quality beyond MDMs.

2 Background

Discrete Diffusion Diffusion is a paradigm for generative modeling where a denoising process p_θ is trained to undo a pre-defined corruption process q [52–54]. Starting from data $\mathbf{x} \sim q_{\text{data}}$, the corruption process produces latent variables $\mathbf{z}_t \sim q(\mathbf{z}_t \mid \mathbf{x})$ for $t \in [0, 1]$, which increasingly move further from the data and towards noise, as t increases.

Adapting diffusion models to discrete data requires a corruption process over the space of sequences of tokens with values in a finite vocabulary [2]. To denote this data, we let $\mathbf{x}, \mathbf{z}_t \in \mathcal{V}$, where $\mathcal{V} := \{0, 1\}^V \subset \Delta^V$, and Δ^V denotes the probability simplex over V categories. We use superscripts to denote the sequence dimension; for example, $\mathbf{x}^{1:L} \in \mathcal{V}^L$ represents a sequence of tokens $(\mathbf{x}^1, \ldots, \mathbf{x}^L)$, where token $\mathbf{x}^\ell \in \mathcal{V}$, for $\ell \in \{1, \ldots, L\}$.

Masked Diffusion Models A promising instantiation of the discrete diffusion paradigm is the recent line of masked diffusion models (MDMs; [33, 37, 46, 51]). In this framework, the corruption process is characterized by marginals that interpolate between data and noise: $q(\mathbf{z}_t \mid \mathbf{x}) = \text{Cat}(\mathbf{z}_t; \alpha_t \mathbf{x} +$

$(1-\alpha_t)\boldsymbol{m}$), where $\boldsymbol{m}$ denotes the one-hot vector for a special `[M]` token and $\alpha_t := \alpha(t) \in [0,1]$ is a noise schedule that is monotonically decreasing in t. In MDM, the corruption process is defined to be 'absorbing,' meaning that once a token transitions to the masked state, it remains in this state.

Diffusion models are trained via a variational bound on the negative log-likelihood (NLL). This bound encourages the learned approximate posterior $p_\theta(\mathbf{z}_s \mid \mathbf{z}_t)$ to match the true one $q(\mathbf{z}_s \mid \mathbf{x}, \mathbf{z}_t)$, for $s < t$ [52]. For MDMs, the true posteriors take the following form [2, 46]:

$$q(\mathbf{z}_s \mid \mathbf{x}, \mathbf{z}_t \neq \boldsymbol{m}) = \mathrm{Cat}(\mathbf{z}_s; \mathbf{z}_t), \quad q(\mathbf{z}_s \mid \mathbf{x}, \mathbf{z}_t = \boldsymbol{m}) = \mathrm{Cat}\left(\mathbf{z}_s; \tfrac{\alpha_s - \alpha_t}{1-\alpha_t}\mathbf{x} + \tfrac{1-\alpha_s}{1-\alpha_t}\boldsymbol{m}\right). \tag{1}$$

A common parameterization for $p_\theta(\mathbf{z}_s \mid \mathbf{z}_t)$ replaces $\mathbf{x}$ in (1) with the output of a neural network: $\mathbf{x}_\theta(\mathbf{z}_t) \in \Delta^V$. Taking the continuous-time limit of the diffusion process, the variational objective for MDMs simplifies to [37, 47, 51]:

$$\mathcal{L}_\theta^{\mathcal{MDM}} := \mathbb{E}_{q_{\text{data}}} \int_0^1 \mathbb{E}_{q_t} \sum_{\ell=1}^{L} \delta_{\mathbf{z}_t^\ell, \boldsymbol{m}} \frac{\dot{\alpha}_t}{1-\alpha_t} \log \langle \mathbf{x}_\theta^\ell(\mathbf{z}_t^{1:L}), \mathbf{x}^\ell \rangle dt, \tag{2}$$

where $\dot{\alpha}_t$ denotes the time derivative of the noise schedule, $\delta_{\mathbf{a},\mathbf{b}} := \delta(\mathbf{a}, \mathbf{b})$ is the Kronecker delta function that evaluates to 1 if $\mathbf{a} = \mathbf{b}$, $\mathbb{E}_{q_{\text{data}}}$ refers to the expectation over drawing data samples $\mathbf{x}^{1:L} \sim q_{\text{data}}$, and $\mathbb{E}_{q_t}$ is the expectation over drawing noisy latents from the forward marginals $\mathbf{z}_t^{1:L} \sim q(\mathbf{z}_t^{1:L} \mid \mathbf{x}^{1:L})$. We optimize this objective via stochastic gradient descent using Monte Carlo samples for the expectations and $t \sim \mathcal{U}[0,1]$ to approximate the integral.

A key drawback of this paradigm is that the denoising network $\mathbf{x}_\theta$ does not learn to modify unmasked inputs $\mathbf{z}_t \neq \boldsymbol{m}$. Without the ability to correct mistakes, during generation, errors propagate and accumulate over time, causing the sampling trajectory to deviate from the true data distribution.

3 Self-Correcting Masked Diffusion Models

In this work, we aim to equip MDMs with the ability to alter previously decoded tokens. Our approach is to have a single model that can act in two 'modes': when inputs contain masked tokens, the model's role is to unmask; when inputs contain all non-mask tokens, the model operates in 'corrector' mode and can update already generated positions.

Below, we present an augmented objective that trains models to jointly unmask and correct mistaken unmasked tokens. Our key insight is that during generation, $\mathbf{x}_\theta$ produces unmasked tokens at each position, some of which align with the true data distribution and some of which potentially contain errors or distributional misalignment. A subset of these tokens are then carried into the next round of iterative refinement. We can interpret outputs of $\mathbf{x}_\theta$ as corrupted sequences from the data distribution, where certain tokens have been replaced by incorrect ones sampled from $\mathbf{x}_\theta$. This perspective motivates our formulation of model-generated sequences serving as inputs for a corrector model.

In Section 3.1, we introduce a straightforward corrector loss that we add to the MDM objective from (2), and we detail a training algorithm to efficiently and jointly train a model to unmask and correct decoding errors. Finally, in Section 4, we provide a sampling algorithm that leverages these abilities to further improve sample quality by interleaving error correction predictions and unmasking steps. Owing to this enhanced ability to iteratively refine any part of the sequence, including already decoded tokens, we dub our method *Pro*gressive *S*elf-*Co*rrection (*ProSeCo*).

3.1 Self-Correcting Objective

The high level idea of our approach is to train a corrector to minimize error between outputs from the denoiser and clean data. By tying weights of the corrector and denoiser networks, we turn this formulation into an augmented MDM variational objective with an auxiliary error correction loss.

More precisely, let $\pi : \Delta^V \to \mathcal{V}$ generate samples from the MDM denoiser model. Then we define the input to our corrector as $\mathbf{y}_t^{1:L} := (\pi \circ \mathbf{x}_\theta^1(\mathbf{z}_t^{1:L}), \ldots \pi \circ \mathbf{x}_\theta^L(\mathbf{z}_t^{1:L}))$. We present the following objective for training a corrector alongside the standard MDM denoiser:

$$\mathcal{L}_{\phi,\theta}^{\mathcal{CMDM}} := \mathbb{E}_{q_{\text{data}}} \int_0^1 \mathbb{E}_{q_t} \sum_{\ell=1}^{L} \Bigg[\underbrace{\lambda_t \log \langle \mathbf{x}_\phi^\ell(\mathbf{y}_t^{1:L}), \mathbf{x}^\ell \rangle}_{\mathcal{L}^{\mathcal{C}}} + \underbrace{\frac{\dot{\alpha}_t}{1-\alpha_t} \delta_{\mathbf{z}_t^\ell, \boldsymbol{m}} \log \langle \mathbf{x}_\theta^\ell(\mathbf{z}_t^{1:L}), \mathbf{x}^\ell \rangle}_{\mathcal{L}^{\mathcal{MDM}}} dt \Bigg], \tag{3}$$

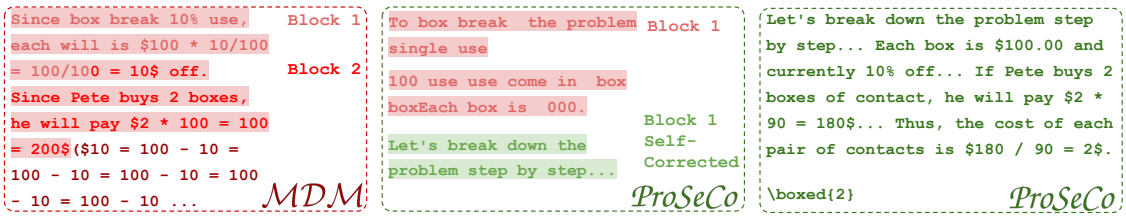

Figure 2: Illustrative example demonstrating the benefits of self-correction. *(Left)* For standard MDM, during parallel unmasking, errors occur. Mistakes accumulate and lead to sample collapse. *(Middle)* For *ProSeCo*, a short correction loop steers generation 'back on track.' *(Right)* *ProSeCo* can thus produce a final high quality output even with parallel generation.

where ϕ denote the parameters of a corrector model $\mathbf{x}_\phi^\ell(\mathbf{y}_t^{1:L}) \in \Delta^V$. The standard MDM loss in the second term ensures that we train a useful denoiser $\mathbf{x}_\theta$ that generates meaningful candidates for $\mathbf{x}_\phi$ to correct. In addition to the MDM objective, we add a correction loss $\mathcal{L}^C$ to every term in the integrand. This auxiliary loss amounts to a simple cross-entropy (CE) term, which encourages the model to identify and correct mistakes from the original denoising network. The weighting $\lambda_t := \lambda(t) < 0$ lets us control the relative weight between the correction and diffusion losses.

Theoretical Motivation The corrector loss term in (3) can also be derived in a principled manner via the lens of learned predictor-corrector samplers for discrete diffusion [30, 67]. In this section, we provide a high-level overview of this argument, deferring the full derivation to Appendix A.

Deviations during sampling result from a mismatch in unconditional marginals: $p_\theta(\mathbf{z}_t) \neq q(\mathbf{z}_t)$. To mitigate this, we apply the following Monte Carlo Markov Chain (MCMC), with learned model p_ϕ:

$$\mathbf{y}_t^{1:L} \sim p_\theta(\mathbf{y}_t^{1:L} \mid \mathbf{z}_t^{1:L}) \quad \longrightarrow \quad \hat{\mathbf{x}}^{1:L} \sim p_\phi(\hat{\mathbf{x}}^{1:L} \mid \mathbf{y}_t^{1:L}) \quad \longrightarrow \quad \hat{\mathbf{z}}_t^{1:L} \sim q(\hat{\mathbf{z}}_t^{1:L} \mid \hat{\mathbf{x}}^{1:L}), \tag{4}$$

with the goal that samples from the chain are ultimately drawn from the correct marginals $q(\mathbf{z}_t)$.

The following condition ensures that the limiting distribution of transition kernels defined by the sampling chain in (4) indeed accomplish this goal (see proof in Appendix A):

$$p_\phi(\mathbf{x}^{1:L} \mid \mathbf{y}_t^{1:L}) \propto p_\theta(\mathbf{y}_t^{1:L} \mid \mathbf{z}_t^{1:L}) q_{\text{data}}(\mathbf{x}^{1:L}). \tag{5}$$

Finally, optimizing ϕ so that the proportionality in (5) holds yields a loss term equivalent to $\mathcal{L}^C$.

3.2 *ProSeCo* Design Decisions

Tying Corrector and Denoiser Weights In order to have a unified model that can both decode and correct, we elect to tie the weights $\phi = \theta$. This decision has the added benefit of eliminating memory overhead for training a separate corrector model. The error correction term in (3) thus becomes a self-correcting one: $\mathcal{L}^C \rightarrow \mathcal{L}^{SC}$

Selecting Transformation π To simplify optimization and ensure a deterministic mapping from $\mathbf{z}_t^{1:L}$ to $\mathbf{y}_t^{1:L}$, we use $\arg\max$ sampling from $\mathbf{x}_\theta(\mathbf{z}_t^{1:L})$. This decision aligns with how state-of-the-art MDMs, such as LLaDA [35], are used in practice. Namely, during generation, at every iteration, each masked position from the output of $\mathbf{x}_\theta$ is decoded greedily. Then using some algorithm or heuristic, we determine which positions to retain for the next round and which to keep masked.

Setting Corrector Loss Weight λ_t We found empirically that reusing the same factor from MDM was a performant strategy, i.e., $\lambda_t = \dot{\alpha}_t/(1-\alpha_t)$. In addition to balancing both terms in our objective, this weighting scheme is justified intuitively. Specifically, in MDM, the weight $\dot{\alpha}_t/(1-\alpha_t)$ discounts samples that are more heavily noised. For training the corrector model, it is

Algorithm 1 *ProSeCo* Training

```
# Differences from standard MDM training highlighted in brown.
```

Input: Training data $\mathcal{D}$, model $\mathbf{x}_\theta$ with parameters θ, corruption process q, noise schedule α_t.
repeat
  Sample $\mathbf{x}^{1:L}$ i.i.d. from $\mathcal{D}$
  Sample $t \sim \mathcal{U}[0,1]$
  Compute $\alpha_t, \dot{\alpha}_t$
  Sample $\mathbf{z}_t^{1:L} \sim q(\mathbf{z}_t^{1:L} \mid \mathbf{x})$
  Compute $\mathbf{x}_\theta(\mathbf{z}_t^{1:L})$
  $\mathcal{L}_\theta^{\mathcal{MDM}} \leftarrow \frac{\dot{\alpha}_t}{1-\alpha_t} \sum_{\ell=1}^L \delta_{\mathbf{z}_t^\ell, \boldsymbol{m}} \log \langle \mathbf{x}_\theta^\ell(\mathbf{z}_t^{1:L}), \mathbf{x}^\ell \rangle$
  $\mathbf{y}_t^\ell \leftarrow \text{sg}(\texttt{one\_hot}(\arg\max_i \mathbf{x}_\theta^\ell(\mathbf{z}_t)_i)), \forall \ell$
  Compute $\mathbf{x}_\theta(\mathbf{y}_t^{1:L})$
  $\mathcal{L}_\theta^{SC} \leftarrow \frac{\dot{\alpha}_t}{1-\alpha_t} \sum_{\ell=1}^L \log \langle \mathbf{x}_\theta^\ell(\mathbf{y}_t^{1:L}), \mathbf{x}^\ell \rangle$
  Perform gradient descent step on $\mathcal{L}_\theta^{\mathcal{MDM}} + \mathcal{L}_\theta^{SC}$
until converged
Return θ

reasonable to apply a similar rationale: sequences with heavy masking which are harder to denoise will also be harder to correct and should therefore be down-weighted.

Combining the above yields our objective for effective joint denoising and self-correction training:

$$\mathcal{L}_\theta^{\mathcal{SCMDM}} := \mathbb{E}_{q_{\text{data}}} \int_0^1 \mathbb{E}_{q_t} \frac{\dot{\alpha}_t}{1-\alpha_t} \sum_{\ell=1}^{L} \left[\underbrace{\log \langle \mathbf{x}_\theta^\ell(\mathbf{y}_t^{1:L}), \mathbf{x}^\ell \rangle}_{\mathcal{L}^{\mathcal{SC}}} + \underbrace{\delta_{\mathbf{z}_t^\ell, \boldsymbol{m}} \log \langle \mathbf{x}_\theta^\ell(\mathbf{z}_t^{1:L}), \mathbf{x}^\ell \rangle}_{\mathcal{L}^{\mathcal{MDM}}} \right] dt. \quad (6)$$

3.2.1 Training with the $\mathcal{SCMDM}$ Objective

In Algorithm 1, we present training for *ProSeCo*, which requires only minor modification to standard MDM training. Specifically, in addition to the MDM forward pass used for (2), we sample from $\mathbf{x}_\theta$, perform a second forward pass, and incorporate the corrector loss. A final modification to (6) is to wrap the denoiser model's outputs in the stop-gradient operation $\text{sg}(\cdot)$ prior to forming the corrector input, which ensures training stability.

4 Sampling with *ProSeCo*

In Algorithm 2, we present sampling for *ProSeCo*. Having a model that can jointly decode and correct allows us to interleave unmasking and correction steps. The goal of corrector iterations is two-fold: they should 1) potentially update already decoded positions in $\mathbf{z}_t^{1:L}$ and 2) provide improved predictions to be used for sampling in the unmasking posterior. To control the computation budget allocated to self-correction, we allow users to specify hyperparameters that determine how often a self-correcting loop is executed, ω, and the number of steps per loop, S.

Algorithm 2 *ProSeCo* Sampling

```
# Differences from standard MDM sampling highlighted in brown.
Input: Model x_θ, length L, unmasking steps T, schedule α_t, self-correction loop budget S, corrector frequency ω.
Initialize z_{t(T)}^{1:L} ← m^{1:L}
for i = T to 1 do
  logits ← x_θ(z_t^{1:L})
  if (T − i + 1) mod ω == 0 then
    z_t^{1:L}, logits ← corrector(x_θ, S, z_t^{1:L}, logits)
    // See function below
  end if
  z_{t(i−1)}^{1:L} ← sample_posterior(logits, z_t^{1:L}, α_{t(i)})
end for
Return sample(x_θ(z_{t(0)}^{1:L}))
function corrector(x_θ, S, z_t^{1:L}, logits)
  Input: Model x_θ, self-correction budget (per step) S, latent sequence z_t^{1:L}, denoising output logits
  Initialize y_t^ℓ ← one_hot(arg max_i logits_i), ∀ℓ
  for S steps do
    corrector_logits ← x_θ(y_t^{1:L})
    y_t^{1:L} ← sample(corrector_logits)
  end for
  # Correct unmasked positions in z_t^{1:L}
  z_t^ℓ ← y_t^ℓ, ∀z_t^ℓ ≠ m
  Return z_t^{1:L}, corrector_logits
end function
```

In the `corrector` routine invoked during sampling, we present the self-correction procedure. When entering this loop, we convert $\mathbf{x}_\theta(\mathbf{z}_t^{1:L})$ into a corrector input via $\arg\max$ sampling at every position. Within each correcting iteration, we use a `sample` method, e.g., greedy-max decoding, to generate the next corrector input sequence from the corrector model outputs. After the inner loop completes, unmasked positions $\mathbf{z}_t^\ell \neq \boldsymbol{m}$ are replaced with corresponding positions in the corrector sequence, which represents the remediation of already decoded tokens.

When operating as an 'unmasking' model, an inference loop iteration is identical to that in standard MDMs: we input a partially masked sequence $\mathbf{z}_t^{1:L}$, we output predictions $\mathbf{x}_\theta^\ell(\mathbf{z}_t^{1:L})$ at every position ℓ, and we decide some tokens to unmask via a call to a `sample_posterior` protocol, which returns a sequence with fewer masked tokens. For example, we can use *ancestral sampling* [47], where we replace $\mathbf{x}$ with $\mathbf{x}_\theta$ in (1) and sample accordingly, or the *confidence-based* approach proposed in Nie et al. [35], where proposal tokens are selected greedily from $\mathbf{x}_\theta(\mathbf{z}_t^{1:L})$ and those with top-k confidence are retained for the next iteration. If correction has been applied, instead of $\mathbf{x}_\theta(\mathbf{z}_t^{1:L})$, we use the corrector model's last output for the `sample_posterior` routine, as the corrector model logits represent better informed predictions.

Table 1: Pass@1 on code and math benchmarks. 'Corrector Sampling' ✗ / ✓ symbol indicates whether a correction algorithm was applied. † indicates values taken from Huang et al. [23]. Other "Prior work" values obtained via evaluation with open-sourced weights. Best value is **bolded**.

			Code		Math	
		Corrector Sampling	HumanEval	MBPP	GSM8K	MATH
Off-the-Shelf 8B Models						
Prior Work	Llama3.1-Instruct [19]	✗	63.41	70.90	81.05	47.38
	LLaDA-Base [35]	✗	34.15	37.20	46.78	17.04
	LLaDA-Instruct [35]	✗	45.73	47.09	83.40	43.14
	+ ReMDM [57]	✓	43.90	45.50	83.93	43.76
	LLaDA1.5 [69]	✗	45.12	46.83	84.00	42.54
	ReMeDi-Instruct† [23]	✓	71.30	57.80	86.30	51.40
Our SFT with LLaDA-Base 8B Model						
Baseline	Vanilla SFT	✗	58.54	56.88	88.86	46.60
	+ ReMDM [57]	✓	56.10	50.00	88.48	46.22
Ours	*ProSeCo* SFT	✗	69.51	57.41	91.36	51.98
	+ *ProSeCo* Max Sampling	✓	**72.56**	**69.31**	**92.19**	**55.06**

5 Experiments

We evaluate *ProSeCo* across a diverse set of tasks and for both conditional and unconditional generation. We consistently demonstrate that our model better scales the quality-efficiency frontier and enables even further improvements via inference-compute scaling.

5.1 Math & Code Benchmarks

Setup To evaluate *ProSeCo* on large MDMs, we perform supervised fine-tuning (SFT) of the LLaDA-Base 8B model [35] using our training Algorithm 1. Specifically, we fine-tune for ∼400B tokens on a modified version of the Llama-Nemotron-Post-Training dataset [5]. In this variant, the target outputs for each prompt were generated using GPT-OSS [36] (see Appendix C.1).

We then evaluate on downstream benchmarks for code: HumanEval [10] and MBPP [3], and math: GSM8K [12] and MATH [21]. In addition to reporting metrics for open-sourced large AR and MDM models (with and without corrector mechanisms), for a direct comparison to our approach, we apply our same SFT recipe using the standard MDM objective to LLaDA-Base.

Note that for all LLaDA-based models we apply the semi-AR sampling algorithm [1] adopted by Nie et al. [35], where the full generation sequence L is broken into blocks of size B and unmasking decoding is applied block-by-block from left-to-right (see Appendix B).

Main Results Table 1 represents our main results. The key finding is that for every benchmark, *ProSeCo* outperforms all diffusion baselines, including those coupled with other corrector mechanisms, and *ProSeCo* beats a comparably-sized instruction fine-tuned AR model (Llama3.1; [19]) on 3 out of 4 tasks. Moreover, our baseline SFT model (fourth row from the bottom) represents a strong watermark, significantly improving over the LLaDA-Base and even surpassing the LLaDA instruction fine-tuned model. Nevertheless, SFT using *ProSeCo* outperforms this strong baseline. Notably, even before applying the *ProSeCo* sampling procedure from Algorithm 2, the model trained with the *ProSeCo* objective outperforms one trained with the standard MDM loss.

Analyzing the Quality-Efficiency Trade-off In Figure 3, we present an analysis of the quality-efficiency trade-offs for *ProSeCo*. Results further to the north-west corner are desirable as they indicate better performance with a smaller number of function evaluations (NFEs).

In standard MDM, the only lever for controlling this trade-off is number of inference steps used for unmasking, or in other words the number of token positions generated in parallel at each decoding step. For *ProSeCo*, we can also control the compute via the frequency of corrector loops and number of iterative refinement steps per loop. For each benchmark, we find that *ProSeCo* can outperform the highest accuracy baseline configuration, i.e., generating one token in each iteration; depicted as the gray dot. As depicted by the green star marker labeled as "Fast," we can strictly improve baseline accuracy with reduced NFEs by increasing the decoding parallelism (4 - 8 tokens) per unmasking

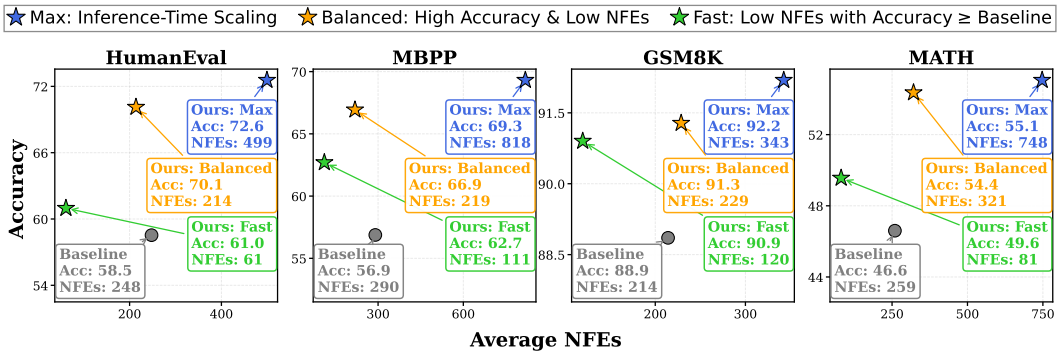

Figure 3: Quality-efficiency trade-offs for *ProSeCo*. Standard MDMs (Baseline; gray dot) attain best performance when decoding a single token every step. *ProSeCo* can vary number of corrector steps and attain comparable performance more efficiently (Ours: Fast; green star), it achieve even better quality for modest increase in compute budget (Ours: Balanced; orange star), or it maximize quality by further scaling inference-time compute (Ours: Max; blue star).

step. To compensate for this, we apply corrector loops at every 2nd decoding iteration and use up to 4 NFEs per corrector loop. Replacing unmasking steps with correcting ones, leads to ∼2-4x speed-ups relative to LLaDA decoding, without sacrificing accuracy. *ProSeCo* also enables configurations that can moderately increase compute while delivering significant accuracy improvements, as depicted by the orange star marker labeled as "Balanced" (for best trade-off; see Appendix C.1 for details on how this point was selected). Finally, *ProSeCo* supports even further scaling of test-time compute to attain our highest performing results depicted in the blue star markers labeled as "Max."

Pareto Frontier for Parallel Decoding and Quality Additionally, in Figure 4, we demonstrate that for standard MDMs, increasing the level of parallel decoding significantly degrades sample quality. In contrast, *ProSeCo* models can recover from the mistakes introduced during generation and better scale the parallel decoding-quality Pareto frontier (see Appendix D.1 for throughput analysis).

Ablation: Selecting Corrector Budget Finally, in Figure 7 (Appendix D.2), we explore the performance of various configurations of our sampling hyperparameters to provide guidance on allocating the corrector budget, as determined by frequency (ω) of and number of steps (S) per loop. The key takeaway, is that our model is highly robust to the choice of these hyperparameters, beating the baseline accuracy at each token parallelism level regardless of choice of ω and S. Additionally for fast sampling regimes (tokens/step $\in \{4, 8\}$), we find that with more frequent corrector loops, we can overcome the drop in quality from parallel decoding and even match or beat the baseline performance achieved with tokens/step $= 1$ with significant speed-up.

Ablation: Standard MDM + Self-Correction To further elucidate the importance of our proposed framework, we apply self-correction sampling from Algorithm 2 to a vanilla SFT model. In Table 6 (Appendix D.4), we see that the standard MDM fails to correct errors, underscoring the core motivation of our work: the baseline model never learns to change already unmasked tokens so its output at unmasked positions is generally uninformative and does not lead to error correction.

5.2 Guided Molecule Design

In the context of guided generation, often when guidance strength is increased, model samples collapse. Our hypothesis is that *ProSeCo* can recover from these errors, thereby improving the guided generation trade-off of maximizing a property of interest while producing a diverse set of samples.

Setup We train models on string representations of molecules known as SMILES [61] from the QM9 dataset [44, 45] (see Appendix C.2). We then apply a discrete classifier-free-guidance (CFG) algorithm [49] with varying unmasking budgets T and guidance strength γ. We measure the number of generated sequences that are valid (can be parsed by RDKit library [29]), unique, and novel (do not appear in the QM9 dataset) as the metric for diverse, high quality samples, and for the novel sequences, we compute the mean property value as the metric for property maximization. We perform this experiment for two properties: ring count and drug-likeness (QED; [6]). We compare

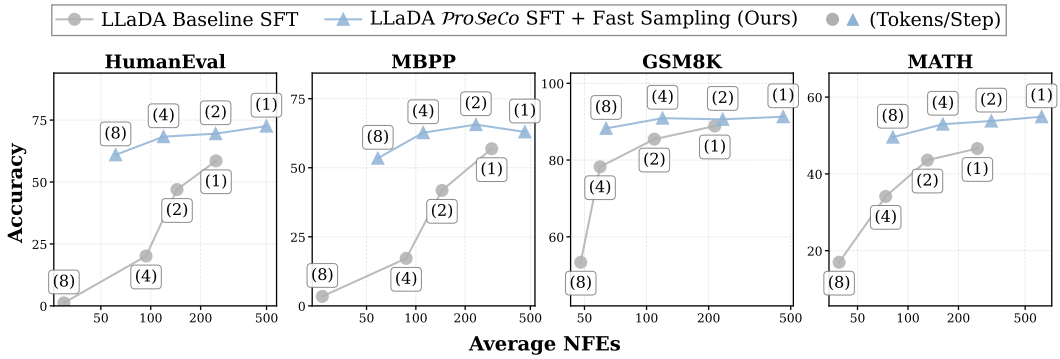

Figure 4: Pareto frontier of parallel decoding and quality. Applying a modest number of corrector steps, allows *ProSeCo* models to recover from paralell decoding errors and extend this frontier.

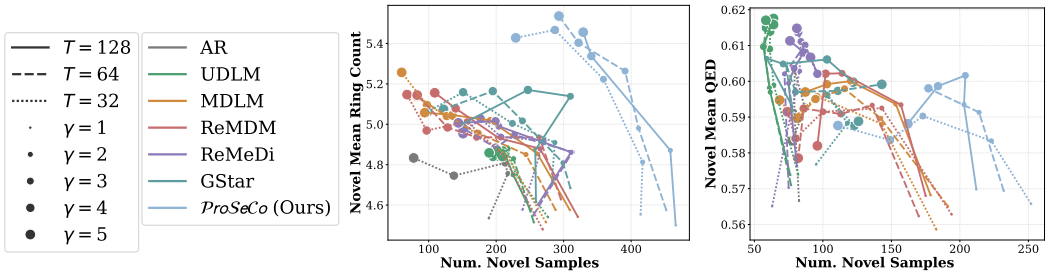

Figure 5: *ProSeCo* better navigates the novelty-property maximization Pareto frontier. Novel samples (x-axis) vs. mean property value of novel samples (y-axis) for classifier-free guidance [49], with varying unmasking steps T (line style) and guidance strength γ (marker size). *(Left)* Maximizing the *ring count* property. *(Right)* Maximizing the *drug likeness (QED)* property.

ProSeCo to an AR model, a diffusion model trained with uniform categorical noise (UDLM; [49]), and a standard masked diffusion model (MDLM; [46]). We also compare to the three remasking strategies ReMDM [57], ReMeDi [23], and GStar [34].

Results In Figure 5, we present the guidance results. Points further north-east are preferable, as they represent property maximization without sacrificing sample diversity and quality. For both properties of interest, *ProSeCo* pushes the Pareto frontier in the desired direction. This is particularly stark for experiments where we maximize the ring count property (left hand side of Figure 5).

Ablation on Choice of λ_t We ablate the effect of our choice of λ_t by repeating the guidance experiment for the ring count property with fixed $\lambda \in \{0.1, 1, 10\}$ and a time-varying weighted $\lambda_t \in \{0.1 \cdot \frac{\dot{\alpha}_t}{(1-\alpha_t)}, 1 \cdot \frac{\dot{\alpha}_t}{(1-\alpha_t)}, 10 \cdot \frac{\dot{\alpha}_t}{(1-\alpha_t)}\}$. We report results in Figure 8 (Appendix D.5), where we find that our design choice of including $\frac{\dot{\alpha}_t}{(1-\alpha_t)}$ in the corrector loss improves performance and that our model is robust to different scaling factors applied to this weighting scheme.

5.3 Unconditional Text Generation

Setup We train *ProSeCo* from scratch on the OpenWebText (OWT; [18]) dataset for 1M steps [46] (see Appendix C.3). We then generate 5000 samples consisting of $L = 1024$ tokens, for varying sampling budgets T. We compute MAUVE [42] and report the perplexity under the GPT2-Large model [43], to also measure quality, and average sequence entropy, to reflect diversity [68]. We compare against an AR model and MDLM [46] without and with correctors: ReMDM [57] and PRISM [25]. We also compare to the hybrid mask and uniform noise model GIDD [56].

For *ProSeCo*, we apply 3 corrector steps after each unmasking step. For parity with the compute budget of other methods, we reduce the number of unmasking steps by a factor of 4 per sampling budget T. For this experiment, we use slightly modified training and sampling (see Appendix C.3).

Table 2: Unconditional generation sample quality for models trained on OpenWebText. † Values reported in Kim et al. [25]. $ Values reported in Wang et al. [57]. * Values obtained with open-sourced weights; sequence length reduced to 512 to match the model trained in von Rütte et al. [56].

	MAUVE (↑)				Gen. PPL (↓)				Entropy (↑)			
Data	1.00				14.8				5.44			
AR ($T = 1024$)$	0.76				12.1				5.22			
$T =$	128	256	512	1024	128	256	512	1024	128	256	512	1024
MDLM$ [46]	0.015	0.023	0.031	0.042	61.5	55.8	53.0	51.3	5.52	5.49	5.48	5.46
ReMDM$ [57]	0.057	0.216	0.350	0.403	42.5	30.5	21.1	28.6	5.43	5.34	5.21	5.38
PRISM† [25]	0.118	0.294	0.423	0.527	21.5	18.0	16.4	15.3	5.18	5.15	5.12	5.10
GIDD* [56]	0.268	0.284	0.334	0.356	95.1	80.5	76.9	76.1	5.00	4.94	4.94	4.94
ProSeCo (Ours)	**0.295**	**0.557**	**0.597**	**0.604**	23.1	16.5	13.2	10.9	5.45	5.39	5.29	5.22

Results In Table 2, we see that across budgets, *ProSeCo* either significantly outperforms or matches baseline methods. Notably, even with just 256 steps, *ProSeCo* attains comparable sample quality to using PRISM with 2x or ReMDM with 4x the inference budget.

6 Related Works

Discrete Diffusion The seminal work of D3PM [2] laid the foundation for adapting diffusion to discrete data. Some works extended this paradigm via the formalism of continuous-time Markov chains [9, 33]. However, our method is more in line with the continuous-time extensions of the variational inference perspective [37, 47, 51]. Previous efforts in this vein have relied on categorical uniform noise corruptions [49, 56] to alleviate the locked-in decoded tokens limitation of MDMs. Recently LLaDA 2.1 [8] scaled up the GIDD framework to 16B-100B parameters by fine-tuning with mixed mask and uniform noise. In contrast, our work which uses the model's own unmasking output to train for correction, more closely aligns training and inference distributions, since during generation mistakes will not resemble uniform noise, but rather will come from the model's own outputs. Our approach thus uses a more 'informed' noise relative to the mask-uniform hybrid approach.

Self-conditioning & Step Unrolling In self-conditioning [11, 14], model predictions of the clean data from one step inform future outputs. Also related to our work is training on unrolled predictions from the denoising trajectory [48], where the model is trained on its own outputs of less noisy latent sequences to more closely simulate the distribution seen during generation. Our method instead uses predictions of clean data, not unrolled trajectories of partially masked sequences.

Corrector Methods Several works used a predictor-corrector framework to improve sample quality, where, following an unmasking predictor step, a corrector step is applied to remask decoded positions [9, 17, 41, 57]. In contrast to these training free methods, other works propose to train an additional head to predict incorrect positions that should be remasked [23, 25, 30, 32, 34]. More related to our work is Zhao et al. [67], which predicts corrections to already decoded tokens. However this method relies on a distinct Hollow Transformer backbone [55]. This severely limits its application to fine-tuning of MDMs, e.g., LLaDA, pre-trained with the standard Transformer backbones.

7 Discussion & Conclusion

In this work, we presented a framework for jointly training a diffusion model to unmask and self-correct. We enable and take advantage of this new ability via minimal and straightforward modifications to standard MDM training and sampling algorithms. Evaluating on conditional and unconditional generation, across various model sizes, we demonstrated that our method consistently outperforms vanilla MDMs and alternative corrector methods both in terms of speed-quality trade-offs and in the ability to further scale inference-time compute for improved generation.

Limitations The key drawback of our work is the added computational cost of the second forward pass during training, especially in contrast to inference-time only schemes, e.g., Wang et al. [57]. However, the empirical results demonstrate that this trade-off of train-time compute is well worth the performance and efficiency gains achieved on downstream evaluations.

Future Directions In follow up work, we plan to explore the disentangling of the corrector and unmasking models via weight untying or with completely separate neural network backbones for each model. Additionally, while we present a performant sampling algorithm, the ability to correct mistakes opens up the design space to more sophisticated schemes of jointly using corrector and unmasking steps, which we leave to future work to explore.

Acknowledgments

This work was partially funded by the National Science Foundation under award CAREER 2145577, and by the National Institute of Health under award MIRA R35GM151243. Marianne Arriola is supported by a NSF Graduate Research Fellowship under award DGE-2139899 and a Hopper-Dean/Bowers CIS Deans Excellence Fellowship.

References

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

[2] Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne Van Den Berg. Structured denoising diffusion models in discrete state-spaces. *Advances in Neural Information Processing Systems*, 34:17981–17993, 2021.

[3] 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*, 2021.

[4] 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.

[5] Akhiad Bercovich, Nir Ailon, Vladimir Anisimov, Tomer Asida, Nave Assaf, Mohammad Dabbah, Ido Galil, Amnon Geifman, Yonatan Geifman, Izhak Golan, et al. Extending puzzle for mixture-of-experts reasoning models with application to gpt-oss acceleration. *arXiv preprint arXiv:2602.11937*, 2026.

[6] G Richard Bickerton, Gaia V Paolini, Jérémy Besnard, Sorel Muresan, and Andrew L Hopkins. Quantifying the chemical beauty of drugs. *Nature chemistry*, 4(2):90–98, 2012.

[7] Tiwei Bie, Maosong Cao, Kun Chen, Lun Du, Mingliang Gong, Zhuochen Gong, Yanmei Gu, Jiaqi Hu, Zenan Huang, Zhenzhong Lan, et al. Llada2. 0: Scaling up diffusion language models to 100b. *arXiv preprint arXiv:2512.15745*, 2025.

[8] Tiwei Bie, Maosong Cao, Xiang Cao, Bingsen Chen, Fuyuan Chen, Kun Chen, Lun Du, Daozhuo Feng, Haibo Feng, Mingliang Gong, et al. Llada2. 1: Speeding up text diffusion via token editing. *arXiv preprint arXiv:2602.08676*, 2026.

[9] 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.

[10] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

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

[12] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[13] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in neural information processing systems*, 34:8780–8794, 2021.

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

[15] William Falcon and The PyTorch Lightning team. PyTorch Lightning, March 2019. URL https://github.com/Lightning-AI/lightning.

[16] Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 12 2023. URL https://zenodo.org/records/10256836.

[17] Itai Gat, Tal Remez, Neta Shaul, Felix Kreuk, Ricky TQ Chen, Gabriel Synnaeve, Yossi Adi, and Yaron Lipman. Discrete flow matching. *arXiv preprint arXiv:2407.15595*, 2024.

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

[19] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[20] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. doi: 10.1038/s41586-020-2649-2. URL https://doi.org/10.1038/s41586-020-2649-2.

[21] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

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

[23] Zemin Huang, Yuhang Wang, Zhiyang Chen, and Guo-Jun Qi. Don't settle too early: Self-reflective remasking for diffusion language models. *arXiv preprint arXiv:2509.23653*, 2025.

[24] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9 (3):90–95, 2007. doi: 10.1109/MCSE.2007.55.

[25] Jaeyeon Kim, Seunggeun Kim, Taekyun Lee, David Z Pan, Hyeji Kim, Sham Kakade, and Sitan Chen. Fine-tuning masked diffusion for provable self-correction. *arXiv preprint arXiv:2510.01384*, 2025.

[26] Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[27] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[28] Volodymyr Kuleshov. Fast algorithms for sparse principal component analysis based on rayleigh quotient iteration. In *International Conference on Machine Learning*, pages 1418–1425. PMLR, 2013.

[29] Greg Landrum et al. Rdkit: A software suite for cheminformatics, computational chemistry, and predictive modeling. *Greg Landrum*, 8(31.10):5281, 2013.

[30] Jose Lezama, Tim Salimans, Lu Jiang, Huiwen Chang, Jonathan Ho, and Irfan Essa. Discrete predictor-corrector diffusion models for image synthesis. In *The Eleventh International Conference on Learning Representations*, 2023.

[31] Xiang Li, John Thickstun, Ishaan Gulrajani, Percy S Liang, and Tatsunori B Hashimoto. Diffusion-lm improves controllable text generation. *Advances in Neural Information Processing Systems*, 35:4328–4343, 2022.

[32] Liming Liu, Binxuan Huang, Xin Liu, Bing Yin, and Tuo Zhao. Teach diffusion language models to learn from their own mistakes. *arXiv preprint arXiv:2601.06428*, 2026.

[33] Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. In *Forty-first International Conference on Machine Learning*, 2024.

[34] Viacheslav Meshchaninov, Egor Shibaev, Artem Makoian, Ivan Klimov, Danil Sheshenya, Andrei Malinin, Nikita Balagansky, Daniil Gavrilov, Aibek Alanov, and Dmitry Vetrov. Guided star-shaped masked diffusion. *arXiv preprint arXiv:2510.08369*, 2025.

[35] 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.

[36] OpenAI. gpt-oss-120b & gpt-oss-20b model card, 2025. URL https://arxiv.org/abs/2508.10925.

[37] 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.

[38] The pandas development team. pandas-dev/pandas: Pandas, February 2020. URL https://doi.org/10.5281/zenodo.3509134.

[39] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[40] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4195–4205, 2023.

[41] 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. *arXiv preprint arXiv:2502.03540*, 2025.

[42] 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.

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

[44] Raghunathan Ramakrishnan, Pavlo O Dral, Matthias Rupp, and O Anatole Von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific data*, 1(1):1–7, 2014.

[45] Lars Ruddigkeit, Ruud Van Deursen, Lorenz C Blum, and Jean-Louis Reymond. Enumeration of 166 billion organic small molecules in the chemical universe database gdb-17. *Journal of chemical information and modeling*, 52(11):2864–2875, 2012.

[46] Subham Sekhar Sahoo, Marianne Arriola, Aaron Gokaslan, Edgar Mariano Marroquin, Alexander M Rush, Yair Schiff, Justin T Chiu, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=L4uaAR4ArM.

[47] Subham Sekhar Sahoo, Aaron Gokaslan, Christopher De Sa, and Volodymyr Kuleshov. Diffusion models with learned adaptive noise. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=loMa99A4p8.

[48] Nikolay Savinov, Junyoung Chung, Mikolaj Binkowski, Erich Elsen, and Aaron van den Oord. Step-unrolled denoising autoencoders for text generation. *arXiv preprint arXiv:2112.06749*, 2021.

[49] Yair Schiff, Subham Sekhar Sahoo, Hao Phung, Guanghan Wang, Sam Boshar, Hugo Dallatorre, Bernardo P de Almeida, Alexander Rush, Thomas Pierrot, and Volodymyr Kuleshov. Simple guidance mechanisms for discrete diffusion models. *arXiv preprint arXiv:2412.10193*, 2024.

[50] Philippe Schwaller, Teodoro Laino, Théophile Gaudin, Peter Bolgar, Christopher A Hunter, Costas Bekas, and Alpha A Lee. Molecular transformer: A model for uncertainty-calibrated chemical reaction prediction. *ACS central science*, 5(9):1572–1583, 2019.

[51] Jiaxin Shi, Kehang Han, Zhe Wang, Arnaud Doucet, and Michalis K Titsias. Simplified and generalized masked diffusion for discrete data. *arXiv preprint arXiv:2406.04329*, 2024.

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

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

[54] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020.

[55] Haoran Sun, Lijun Yu, Bo Dai, Dale Schuurmans, and Hanjun Dai. Score-based continuous-time discrete diffusion models. *arXiv preprint arXiv:2211.16750*, 2022.

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

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

[58] Guanghan Wang, Yair Schiff, Gilad Turok, and Volodymyr Kuleshov. d2: Improved techniques for training reasoning diffusion language models. *arXiv preprint arXiv:2509.21474*, 2025.

[59] Yingheng Wang, Yair Schiff, Aaron Gokaslan, Weishen Pan, Fei Wang, Christopher De Sa, and Volodymyr Kuleshov. InfoDiffusion: Representation learning using information maximizing diffusion models. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 36336–36354. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/wang23ah.html.

[60] Michael L. Waskom. seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60):3021, 2021. doi: 10.21105/joss.03021. URL https://doi.org/10.21105/joss.03021.

[61] David Weininger. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of chemical information and computer sciences*, 28 (1):31–36, 1988.

[62] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

[63] Chengyue Wu, Hao Zhang, Shuchen Xue, Zhijian Liu, Shizhe Diao, Ligeng Zhu, Ping Luo, Song Han, and Enze Xie. Fast-dllm: Training-free acceleration of diffusion llm by enabling kv cache and parallel decoding. *arXiv preprint arXiv:2505.22618*, 2025.

[64] Omry Yadan. Hydra - a framework for elegantly configuring complex applications. Github, 2019. URL https://github.com/facebookresearch/hydra.

[65] 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.

[66] Lingxiao Zhao, Xueying Ding, Lijun Yu, and Leman Akoglu. Improving and unifying discrete&continuous-time discrete denoising diffusion. *arXiv preprint arXiv:2402.03701*, 2024.

[67] Yixiu Zhao, Jiaxin Shi, Lester Mackey, and Scott Linderman. Informed correctors for discrete diffusion models. *arXiv preprint arXiv:2407.21243*, 2024.

[68] 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.

[69] Fengqi Zhu, Rongzhen Wang, Shen Nie, Xiaolu Zhang, Chunwei Wu, Jun Hu, Jun Zhou, Jianfei Chen, Yankai Lin, Ji-Rong Wen, et al. Llada 1.5: Variance-reduced preference optimization for large language diffusion models. *arXiv preprint arXiv:2505.19223*, 2025.

Contents

A Deriving *ProSeCo* from MCMC Predictor-Corrector

We now show how we can derive the corrector loss term $\mathcal{L}^C$ from the objective in (3) by leveraging formulations from predictor-corrector sampling for continuous-time Markov chains [9]. In particular, we start from the perspective that sampling errors are a byproduct of the fact that the posteriors of the learned reverse process do not align with those from predefined corruption process $p_\theta(\mathbf{z}_s \mid \mathbf{z}_t) \neq q(\mathbf{z}_s \mid \mathbf{z}_t)$. This issue arises because of two reasons, 1) the model may not fully capture the exact marginals of each token position, and 2) the inherent parameterization of our reverse process as factorized independently across the sequence length dimension does not account for cross-token dependencies [4]. These deviations compound along a generation path and ultimately lead to marginals that are significantly divergent $p_\theta(\mathbf{z}_t) \neq q(\mathbf{z}_t)$. In score-matching generative modeling for continuous data [54], this mismatch can be mitigated by Langevin Monte Carlo Markov chain (MCMC) iterations. Unfortunately in the discrete data setting, we do not have access to a continuous score function that drives the predictor-corrector sampler proposed in Song et al. [54].

Crucially, Campbell et al. [9] demonstrate that for discrete data we can also apply corrector steps and produce a MCMC that yields $q(\mathbf{z}_t)$ as its stationary distribution. Interleaving corrector MCMC steps with predictor (unmasking) ones enables us to recover from deviations before proceeding along the generation trajectory. While Campbell et al. [9] observe better sample quality with their proposed 'forward-backward' corrector scheme, follow up works [30, 66] note that this methodology can be further improved by learning the reverse transition kernel for the corrector MCMC, leading to more 'informed' corrector steps.

At a high level, our derivation below proceeds as follows. We first propose a corrector sampling chain. We then prove that if the corrector distribution satisfies a given functional form then we will achieve our goal of having the samples from the MCMC corrector step $\sim q(\mathbf{z}_t^{1:L})$. Finally, we enforce this condition by minimizing a divergence between the corrector distribution induced by our model and the desired form. This training recovers the corrector loss proposed in Section 3.

Corrector MCMC In the vein of learned predictor-corrector works [30], we define an MCMC corrector step by the following chain:

$$\begin{aligned} &\text{1) Sample } \mathbf{y}_t^{1:L} \sim p_\theta(\mathbf{y}_t^{1:L} \mid \mathbf{z}_t^{1:L}), \\ &\text{2) Sample } \hat{\mathbf{x}}^{1:L} \sim p_\phi(\hat{\mathbf{x}}^{1:L} \mid \mathbf{y}_t^{1:L}), \\ &\text{3) Sample } \hat{\mathbf{z}}_t^{1:L} \sim q(\hat{\mathbf{z}}_t^{1:L} \mid \hat{\mathbf{x}}^{1:L}), \end{aligned} \tag{7}$$

where $p_\theta(\mathbf{y}_t^{1:L} \mid \mathbf{z}_t^{1:L}) = \prod_{\ell=1}^{L} \mathrm{Cat}(\mathbf{y}_t^\ell; \mathbf{x}_\theta^\ell(\mathbf{z}_t^{1:L}))$ is the distribution defined by parallel decoding from the output of our denoising network $\mathbf{x}_\theta$ and $q(\hat{\mathbf{z}}_t^{1:L} \mid \hat{\mathbf{x}}^{1:L})$ follows the factorized masked diffusion forward process, introduced in Section 2. Additionally, p_ϕ represents a corrector model with parameters ϕ.

Intuitively the sampling chain above represents the following correction mechanism: 1) we sample from the denoising model's prediction given the current latent variable, $\mathbf{y}_t^{1:L} \sim p_\theta(\mathbf{y}_t^{1:L} \mid \mathbf{z}_t^{1:L})$, then 2) given this sample, the corrector predicts the clean data distirbution, and we sample from this prediction, $\hat{\mathbf{x}}^{1:L} \sim p_\phi(\hat{\mathbf{x}}^{1:L} \mid \mathbf{y}_t^{1:L})$, and finally 3) we re-noise the sample from the corrector output according to the predefined noising process, $\hat{\mathbf{z}}_t^{1:L} \sim q(\hat{\mathbf{z}}_t^{1:L} \mid \hat{\mathbf{x}}^{1:L})$. The goal of this chain is that the re-noised latent variable $\hat{\mathbf{z}}_t^{1:L}$ more closely resembles draws from the true distribution $q(\mathbf{z}_t^{1:L})$.

Detailed Balance Condition We now describe a condition on p_ϕ that will achieve this goal.

Proposition A.1. *The following form will ensure the detailed balanced condition, with $q(\mathbf{z}_t^{1:L})$ as the stationary distribution:*

$$p_\phi(\mathbf{x}^{1:L} \mid \mathbf{y}_t^{1:L}) q(\mathbf{z}_t^{1:L} \mid \mathbf{x}^{1:L}) = \frac{p_\theta(\mathbf{y}_t^{1:L} \mid \mathbf{z}_t^{1:L}) q(\mathbf{z}_t^{1:L} \mid \mathbf{x}^{1:L}) q_{data}(\mathbf{x}^{1:L})}{Z(\mathbf{y}_t^{1:L})}, \tag{8}$$

where $Z(\mathbf{y}_t^{1:L})$ is a normalization constant representing the marginal probability of $\mathbf{y}_t^{1:L}$.

Proof. We prove this using a similar argument to that presented in Lezama et al. [30]. Specifically, let $R(\hat{\mathbf{z}}_t^{1:L} \mid \mathbf{z}_t^{1:L})$ denote the transition kernel defined by the sampling chain in (7), then

$$q(\mathbf{z}_t^{1:L}) R(\hat{\mathbf{z}}_t^{1:L} \mid \mathbf{z}_t^{1:L}) = q(\mathbf{z}_t^{1:L}) \sum_{\mathbf{y}_t^{1:L}} \sum_{\mathbf{x}^{1:L}} p_\phi(\mathbf{x}^{1:L} \mid \mathbf{y}_t^{1:L}) q(\hat{\mathbf{z}}_t^{1:L} \mid \mathbf{x}^{1:L}) p_\theta(\mathbf{y}_t^{1:L} \mid \mathbf{z}_t^{1:L}) \quad \textit{\{From (7)\}}$$

$$= q(\mathbf{z}_t^{1:L}) \sum_{\mathbf{y}_t^{1:L}} \sum_{\mathbf{x}^{1:L}} \left(\frac{p_\theta(\mathbf{y}_t^{1:L} \mid \hat{\mathbf{z}}_t^{1:L}) q(\hat{\mathbf{z}}_t^{1:L} \mid \mathbf{x}^{1:L}) q_{\text{data}}(\mathbf{x}^{1:L})}{Z(\mathbf{y}_t^{1:L})} \right) p_\theta(\mathbf{y}_t^{1:L} \mid \mathbf{z}_t^{1:L}) \quad \textit{\{From (8)\}}$$

$$= q(\mathbf{z}_t^{1:L}) \sum_{\mathbf{y}_t^{1:L}} \left(\frac{p_\theta(\mathbf{y}_t^{1:L} \mid \hat{\mathbf{z}}_t^{1:L}) q(\hat{\mathbf{z}}_t^{1:L})}{Z(\mathbf{y}_t^{1:L})} \right) p_\theta(\mathbf{y}_t^{1:L} \mid \mathbf{z}_t^{1:L}) \quad \textit{\{Marginalize } \mathbf{x}\textit{\}}$$

$$= q(\hat{\mathbf{z}}_t^{1:L}) \sum_{\mathbf{y}_t^{1:L}} \left(\frac{p_\theta(\mathbf{y}_t^{1:L} \mid \mathbf{z}_t^{1:L}) q(\mathbf{z}_t^{1:L})}{Z(\mathbf{y}_t^{1:L})} \right) p_\theta(\mathbf{y}_t^{1:L} \mid \hat{\mathbf{z}}_t^{1:L}) \quad \textit{\{Note 'role change' of } \mathbf{z}_t \textit{ \& } \hat{\mathbf{z}}_t\textit{\}}$$

$$= q(\hat{\mathbf{z}}_t^{1:L}) \sum_{\mathbf{y}_t^{1:L}} \sum_{\mathbf{x}^{1:L}} \left(\frac{p_\theta(\mathbf{y}_t^{1:L} \mid \mathbf{z}_t^{1:L}) q(\mathbf{z}_t^{1:L} \mid \mathbf{x}^{1:L}) q_{\text{data}}(\mathbf{x}^{1:L})}{Z(\mathbf{y}_t^{1:L})} \right) p_\theta(\mathbf{y}_t^{1:L} \mid \hat{\mathbf{z}}_t^{1:L})$$

$$= q(\hat{\mathbf{z}}_t^{1:L}) \sum_{\mathbf{y}_t^{1:L}} \sum_{\mathbf{x}^{1:L}} p_\phi(\mathbf{x}^{1:L} \mid \mathbf{y}_t^{1:L}) q(\mathbf{z}_t^{1:L} \mid \mathbf{x}^{1:L}) p_\theta(\mathbf{y}_t^{1:L} \mid \hat{\mathbf{z}}_t^{1:L}) \quad \textit{\{From (8)\}}$$

$$= q(\hat{\mathbf{z}}_t^{1:L}) R(\mathbf{z}_t^{1:L} \mid \hat{\mathbf{z}}_t^{1:L}). \quad \textit{\{From (7)\}}$$

□

Maximum Likelihood Corrector Training Having shown that the condition in (8) is sufficient for achieving the desired limiting distribution $q(\mathbf{z}_t^{1:L})$ for the corrector MCMC, we can cast the corrector training as ensuring that $p_\phi(\mathbf{x}^{1:L} \mid \mathbf{y}_t^{1:L}) q(\mathbf{z}_t^{1:L} \mid \mathbf{x}^{1:L}) \propto p_\theta(\mathbf{y}_t^{1:L} \mid \mathbf{z}_t^{1:L}) q(\mathbf{z}_t^{1:L} \mid \mathbf{x}^{1:L}) q_{\text{data}}(\mathbf{x}^{1:L})$ [30]. We can achieve this by minimizing the following objective:

$$\begin{aligned} &D_{\text{KL}}[p_\theta(\mathbf{y}_t^{1:L} \mid \mathbf{z}_t^{1:L}) q(\mathbf{z}_t^{1:L} \mid \mathbf{x}^{1:L}) q_{\text{data}}(\mathbf{x}^{1:L}) || p_\phi(\mathbf{x}^{1:L} \mid \mathbf{y}_t^{1:L}) q(\mathbf{z}_t^{1:L} \mid \mathbf{x}^{1:L}) Z(\mathbf{y}_t^{1:L})] \\ &\quad = -\mathbb{E}_{\mathbf{x}^{1:L} \sim q_{\text{data}}(\mathbf{x}^{1:L})} \mathbb{E}_{\mathbf{z}_t^{1:L} \sim q(\mathbf{z}_t^{1:L} \mid \mathbf{x}^{1:L})} \mathbb{E}_{\mathbf{y}_t^{1:L} \sim p_\theta(\mathbf{y}_t^{1:L} \mid \mathbf{z}_t^{1:L})} [\log p_\phi(\mathbf{x}^{1:L} \mid \mathbf{y}_t^{1:L})] + C, \end{aligned} \tag{9}$$

where C represents factors that do not depend on ϕ. Minimizing this objective ensures that the distribution over latents $\mathbf{z}_t^{1:L}$ induced by the corrector steps aligns with the true marginal distributions.

Parameterizing the Corrector Model Finally, similar to the denoising model, we parameterize the corrector to be independent across the sequence length dimension given its inputs, as follows:

$$p_\phi(\mathbf{x}^{1:L} \mid \mathbf{y}_t^{1:L}) = \prod_{\ell=1}^{L} p_\phi(\mathbf{x}^\ell \mid \mathbf{y}_t^{1:L}), \quad \text{with } p_\phi(\mathbf{x}^\ell \mid \mathbf{y}_t^{1:L}) = \text{Cat}(\mathbf{x}^\ell; \mathbf{x}_\phi^\ell(\mathbf{y}_t^{1:L})), \tag{10}$$

where $\mathbf{x}_\phi^\ell(\mathbf{z}_t^{1:L}) \in \Delta^V$ for all $\ell \in \{1, \ldots, L\}$ is the output of a corrector model network. This independent factorization for the corrector model enables us to maintain parallel generation.

Using this factorized parameterization and replacing the sampling of $\mathbf{y}_t^{1:L}$ with the transformation $\pi \circ \mathbf{x}_\theta^{1:L}$ defined in Section 3.1 recovers the $\mathcal{L}^C$ corrector loss term from the objective in (3):

$$\mathcal{L}^C = \log \langle \mathbf{x}_\phi^\ell(\mathbf{y}_t^{1:L}), \mathbf{x}^\ell \rangle.$$

B Sampling with *ProSeCo* Semi-Autoregressively

In Algorithm 3, we present a modified version of our sampling method from Algorithm 2, which accommodates the block autoregressive decoding proposed in Arriola et al. [1] and adopted by LLaDA [35]. Given that we applied block AR decoding to the LLaDA models, the implementation provided in Algorithm 3 assumes full bidirectional attention is applied across the entire sequence at every forward pass, as in LLaDA, and is not written to support key-value (KV) caching. However, this algorithm can be easily adapted to support efficient KV caching proposed in Arriola et al. [1].

Notably, for *ProSeCo* with semi-AR decoding, at every correction iteration, clean tokens in the current block and all previously decoded blocks can be adapted.

C Additional Experimental Details

C.1 Math & Code Benchmarks

Dataset Our SFT dataset is a modified version of the Llama-Nemotron-Post-Training dataset [5], comprising approximately 32 million samples across four core domains: mathematical reasoning,

Algorithm 3 *ProSeCo* Sampling Block Autoregressive

```
# Assumes full bidirectional attention without KV-caching, as in LLaDA.
# Differences to standard MDM with block AR decoding highlighted in brown.
```

Input: Model $\mathbf{x}_\theta$, length L, block size B, unmasking steps T, noise schedule α_t, self-correction budget (per step) S, corrector frequency ω.
Initialize $\mathbf{z}_{t(T)}^{1:L} \leftarrow \boldsymbol{m}^{1:L}$
for $b = 1$ to (L/B) **do**
  for $i = T$ **to** 1 **do**
    $\texttt{logits} \leftarrow \mathbf{x}_\theta(\mathbf{z}_t^{1:L})$
    if $(T - i + 1) \mod \omega == 0$ **then**
      $\mathbf{z}_t^{1:L}, \texttt{logits} \leftarrow \texttt{corrector}(\mathbf{x}_\theta, \mathbf{z}_t^{1:L}, S)$
    end if
    $\texttt{logits}^\ell \leftarrow -\infty, \forall \ell \in [1, (b-1) \cdot B] \cup [b \cdot B + 1, L]$
    $\mathbf{z}_{t(i-1)}^{1:L} \leftarrow \texttt{sample\_posterior}(\texttt{logits}, \mathbf{z}_t^{1:L}, \alpha_{t(i)})$
  end for
  $\mathbf{z}_{t(0)}^{\ell} \leftarrow \texttt{sample}(\mathbf{x}_\theta^\ell(\mathbf{z}_{t(0)}^{1:L})), \forall \ell \in [1 + (b-1) \cdot B, b \cdot B]$
  $\mathbf{z}_{t(T)}^{1:L} \leftarrow \mathbf{z}_{t(0)}^{1:L}$
end for
Return $\mathbf{z}_{t(0)}^{1:L}$

coding, science, and instruction following. The distribution is heavily skewed toward math (66.84%) and coding (30.62%), with smaller allocations for science (2.15%), instruction following (0.17%), chat (0.12%), and safety (0.10%). All target outputs were generated using GPT-OSS [36] in high-reasoning mode.

Training Hyperparameters We fine-tuned the LLaDA-Base 8B model for ∼400 B tokens, which amounts to 3 epochs of training on our SFT dataset. We train with a batch size of 1024. For learning rate we linearly warm-up for 1000 gradient steps until a maximum learning rate of 2.5e^{-5}. After this peak, we apply cosine decay until a minimum learning rate of 2.5e^{-7}. We use the ADAMW optimizer [26] with beta parameters (0.9, 0.999). Finally, during training it is common to set a $\texttt{min\_t} > 0$ value which biases the sampling of timesteps away from uniform over the unit interval, by shifting all samples to be in the range $[\texttt{min\_t}, 1]$. For example, in works such as Sahoo et al. [47], this value is set to 1e^{-3}. For our SFT experiments, we found that biasing towards heavier masking during training led to improved performance, hence we set $\texttt{min\_t}$ to 1e^{-1}.

Compute resources We fine-tuned both our model and the baseline using 256 NVIDIA B200 GPUs, and conducted all evaluation runs on 8 NVIDIA H100 GPUs.

Evaluation For evaluation, we rely on the Nemo Skills library. We evaluate all models with batch size 1. This is to mitigate varying padding lengths based on prompt size variation and to enable effective use of early-stopping whenever the `[EOS]` token is generated. All evaluated models use a maximum generation length of 1024.

We evaluate 4 benchmarks: HumanEval [10] and MBPP [3], GSM8K [12] and MATH [21]. We use 0-shot evaluation for each benchmark.

For LLaDA models, we use a semi-AR decoding scheme [1], as in Nie et al. [35], with default block size of 32.

Prompts and Evaluation Templates Below, we detail the templates used for evaluating models in Table 1. These prompts were used for all models, except the off-the-shelf LLaDA-Base model. For HumanEval and MBPP the placeholder `{question}` is replaced by the prompt from the benchmark dataset. Similarly, for GSM8K and MATH, the placeholder `{problem}` is replaced by the prompt from the benchmark dataset. Additionally, we use the default chat templates from the Nemo Skills library.

Note that for LLaDA-Base, we do not provide any additional prompt text, we only use the benchmark question as given.

Sampling Hyperparameters When evaluating *ProSeCo* models, we explore different configurations of unmasking and corrector budgets. In Table 3, we detail the sampling hyperparameters used to generate the best performing results for *ProSeCo* Max reported in Table 1 and Figure 3; see Ap-

HumanEval / MBPP Prompt:	Here is a problem for which you need to generate/complete code:
	{question}
	
	Please continue to complete the function with python programming language. You are not allowed to modify the given code and do the completion only.
	The solution should be in the following format:
	``python
	# Your code here
	``

GSM8K / MATH Prompt:	Solve the following math problem. Show your step-by-step reasoning, then provide the final answer inside \boxed{} {problem}

pendix D.3 for more details. Note that T represents a maximum unmasking budget, since we apply early stopping on the [EOS] token. Additionally, S represents a maximum corrector budget per correction loop, because we break the loop iterations if a corrector sequence does not change between rounds.

For Figure 4 (and Figure 6, below), we use use the Fast configuration, applying a corrector loop every 4 unmasking steps, with a maximum of 2 corrector steps per loop.

Table 3: Sampling hyperparameters for results attained with *ProSeCo* and reported in Figure 3.

	HumanEval	**MBPP**	**GSM8K**	**MATH**
Baseline (Vanilla SFT)				
Accuracy (%)	58.54	56.88	88.86	46.60
Average NFEs	247.8	290.3	214.2	259.4
Generation length	1024	1024	1024	1024
Block length	32	32	32	32
Demasking Tokens/Step	1	1	1	1
ProSeCo Max				
Accuracy (%)	72.56	69.31	92.19	55.06
Average NFEs	499.2	818.3	342.5	747.5
Generation length	1024	1024	1024	1024
Block length	32	32	32	32
Demasking Tokens/Step	1	1	1	1
Corrector frequency ω	2	1	2	2
Maximum corrector steps per loop S	4	8	1	8
ProSeCo Balanced				
Accuracy (%)	70.12	66.93	91.28	54.36
Average NFEs	213.9	218.7	228.7	321.2
Generation length	1024	1024	1024	1024
Block length	32	32	32	32
Demasking Tokens/Step	4	4	4	4
Corrector frequency ω	1	1	1	1
Maximum corrector steps per loop S	8	8	8	8
ProSeCo Fast				
Accuracy (%)	60.98	62.70	90.90	49.56
Average NFEs	61.4	110.9	119.5	81.2
Generation length	1024	1024	1024	1024
Block length	32	32	32	32
Demasking Tokens/Step	8	4	4	8
Corrector frequency ω	2	2	2	2
Maximum corrector steps per loop S	4	4	4	4

Baselines For all prior work results, other than ReMeDi [23], we use open-source weights and evaluate using with batch size 1, maximum generation length of 1024, and early stopping on the

[EOS] token. For ReMeDi Instruct, we report values from Huang et al. [23], as we were unable to reproduce or improve upon their results using the open-source model provided.

All model evaluations, other than for the off-the-shelf LLaDA-Base model, were performed using the Nemo Skills library, using default chat templates from this library and the prompt templates described above. For LLaDA-Base, since the Nemo Skills repository injects chat templates by default, we evaluate using the lm-eval-harness repository.

For ReMDM, we follow the algorithm proposed by Wang et al. [57] for applying this method to LLaDA. Namely, for each block of 32 tokes, once 28 tokens have been generated, we enable a ReMDM loop, where for 32 iterations we remask 2 tokens that had the lowest confidence at the time at which they were decoded and unmask 2 tokens based on their confidence. Hence, at the end of the ReMDM loop, there are still 28 unmasked and 4 masked tokens, at which point we finish generating using the standard LLaDA confidence-based sampling.

C.2 Guided Molecule Design

For this experiment, we follow the setup detailed in Schiff et al. [49].

Dataset We train on the QM9 dataset [44, 45], which consists of $\sim$133k molecules represented as SMILES strings [61]. We use the RDKit library [29] to add the ring count and drug likeness (QED; [6]) annotations. The dataset was tokenized using a regular expression tokenizer [50]. We use sequence length of $L = 32$, with right-padding.

For each property, we generate binary labels that indicate whether a sample is below or above the 90th percentile of training samples. For discrete classifier-free-guidance [49], we train with this label for conditional models, and randomly 'drop it out' 10 percent of the time by replacing it with a 'masked' label to simulate unconditional modeling.

Hyperparameters Hyperparameters follow Schiff et al. [49]. Namely, we use a DiT-style [40] backbone with 92.4M parameters. Models were trained with a batch size 2048, and we perform 25k gradient updates. We use a maximum learning rate of $3\mathrm{e}^{-4}$ that we linearly warm-up to for 1000 steps. After this peak we apply cosine decay until a minimum learning rate of $3\mathrm{e}^{-6}$. We use the ADAM-W with beta parameters $(0.9, 0.999)$.

Of note, when training *ProSeCo* models for ring count, we found it beneficial to eliminate the 'copy over' parameterization of the denoising network $\mathbf{x}_\theta$ proposed in Sahoo et al. [46]. That is, we do not enforce that $\mathbf{x}_\theta$ simply copy over any token positions $\mathbf{z}_t^\ell \neq \boldsymbol{m}$. For training models for the QED property, we maintained this copy-over parameterization.

Compute Resources We trained our model on 8 A5000 GPUs.

Evaluation We generate 1024 samples from our model using various unmasking budgets T and guidance temperature γ. Of note, when applying the corrector model forward passes, we only use the conditional model, i.e., $\gamma = 1$.

We use the RDKit library to parse generated samples. Of the valid strings (those that can be parsed) we retain unique samples that are not found in the original QM9 dataset (novel). We then use RDKit to measure the property of interest for these novel samples.

Sampling Hyperparameters For both ring count and QED maximization we use $\omega = T/2$ for corrector loop frequency and $S = T/16$ for steps per loop. Evaluation was performed with an exponential moving average (EMA) checkpoint. During training we used an EMA decay factor of 0.9999.

Baselines Values for AR, UDLM, MDLM, and ReMDM were taken from Wang et al. [57].

For ReMeDi [23] and GStar [34], we re-implement these methods and train them on the QM9 dataset. Both of these methods entail training a classifier that predicts incorrect tokens to remask.

For ReMeDi, we jointly train their "Unmasking Policy Stream" (UPS) alongside an MDLM model. We use the "dual stream" architecture recommended by Huang et al. [23], with 4 DiT layers for the UPS, connected to the main backbone at layers 0, 3, 7, and 10. We set the UPS loss weight at $\lambda_{\text{UPS}} = 0.3$. For the proportion of unmasked tokens that get flipped to random tokens we use $\rho_{t,\text{incorrect}} = 0.1$. Finally, as ReMeDi was not originally proposed within th context of classifier-free guidance, to have

a fair comparison, we perform a hyperparameter sweep for this model over the following parameters: whether to apply the classifier-free guidance (CFG) to only the "Token Prediction Stream" (i.e., the MDLM output) or also to the UPS and whether to use sampling or greedy-max selection when choosing which tokens to remask. We found best results when applying CFG to both streams and when using proportional sampling, not greedy-max selection, for remasking positions.

In GStar, the error prediction head is trained with a frozen diffusion backbone model, hence we use the pre-trained MDLM from Wang et al. [57] and freeze its weights. The error prediction head consists of a `LayerNorm` and linear projection from the final hidden token representations of the frozen backbone. We employ the "GStar+" sampling algorithm and for fair comparison we perform a sweep over the following parameters: whether to apply the CFG to only the MDLM output or also to the remasking prediction head, whether to use Gumbel top-k or greedy top-k selection during remasking, the temperature for both the MDLM (sweep over $\{0.7, 1.0, 1.3\}$) and remasking head output (sweep over $\{2, 4, 8, 16\}$), and the remasking switch-on time (sweep over $\{0.1, 0.2, 0.3, 0.4\}$). For the ring count property, we found best results using Gumbel top-k, switch time 0.2, denoiser temperature 0.7, and prediction head temperature 16. For the QED property, we found best results using greedy top-k, switch time 0.4, and denoiser temperature 1.0.

C.3 Unconditional Text Generation

For this experiment, we follow the setup described in Sahoo et al. [46].

Dataset We train models on the OpenWebText (OWT; [18]) dataset. We tokenized using the `gpt-2` [43] tokenizer and created sequences of $L = 1024$ tokens by wrapping samples and separating them with an `[EOS]` token. We also also place an `[EOS]` token at the beginning and end of each sequence.

Hyperparameters As in Sahoo et al. [46], we use a DiT backbone with 170M parameters. We used a batch size of 512 and applied 1M gradient updates. We use a constant learning rate of $3\mathrm{e}^{-4}$ that we linearly warm-up to for 2500 steps. We use the ADAM-W optimizer with beta parameters (0.9, 0.999). As described in Appendix C.1, we use a `min_t` value of $1\mathrm{e}^{-1}$, when training *ProSeCo* models on OWT.

For training *ProSeCo* in this setting, we found it beneficial to eliminate the 'copy over' parameterization of the denoising network $\mathbf{x}_\theta$.

Compute Resources We trained our model on 32 H100 GPUs.

Evaluation We follow the evaluation protocol from Wang et al. [57]. Specifically, we generate 5000 samples and compute the MAUVE metric [42], generative perplexity under the `gpt2-large` model, and entropy of generated tokens.

Modified Training Objective For this experiment, we use a slightly modified objective relative to that presented in Algorithm 1. Specifically, we adjust the self-correction term $\mathcal{L}^{\mathcal{SC}}$ so that gradients for tokens which the original denoiser model incorrectly predicts are steeper. To accomplish this, we replace the self-correction loss term from Algorithm 1 with the following:

$$\tilde{\mathcal{L}}^{\mathcal{SC}}(\theta) := \frac{\dot{\alpha}_t}{1-\alpha_t} \sum_{\ell=1}^{L} \delta_{\mathbf{y}_t^\ell, \mathbf{x}^\ell} \langle \mathbf{x}_\theta^\ell(\mathbf{y}_t^{1:L}), \mathbf{x}^\ell \rangle + (1 - \delta_{\mathbf{y}_t^\ell, \mathbf{x}^\ell}) \log \langle \mathbf{x}_\theta^\ell(\mathbf{y}_t^{1:L}), \mathbf{x}^\ell \rangle. \tag{11}$$

The modified loss term in (11) has the desirable property that token positions where the original denoiser model predicts a mistake, i.e. $\delta_{\mathbf{y}_t^\ell, \mathbf{x}^\ell} = 0$, are prioritized. This is accomplished by the fact that the gradient for the standard cross-entropy loss used for these token positions is steeper than that for positions where the denoising model is 'already correct', i.e., $\delta_{\mathbf{y}_t^\ell, \mathbf{x}^\ell} = 1$, where the loss is taken without the logarithm.

Modified Sampling Procedure Additionally, for the samples presented in Table 2, we use a slightly modified sampling procedure. Rather than simply using $\arg\max$ decoding at every token position for the `sample` sub-routine in the `corrector` function in Algorithm 2, at each corrector step, we sort all positions by the confidence of the $\arg\max$ token. Then, for the top-k positions, we use the $\arg\max$ token, and for the remaining positions, we leave them unchanged. We use $k = 100$ and found that this procedure better encouraged sample diversity and improved MAUVE scores. In Algorithm 4, we detail the top-k sampling sub-routine that we use for corrector sampling in the OWT experiments.

Algorithm 4 Corrector `sample` sub-routine used in OWT experiments

Input: Corrector model input $\mathbf{y}_t^{1:L}$, corrector model output `corrector_logits`, parameter k
$\hat{\mathbf{y}}^{1:L} \leftarrow \arg\max(\texttt{corrector\_logits})$
`# 'gather' is an API that returns values of its first input at the indices specified by its second input, see for example torch.gather (with dim=-1)`
$\texttt{confidence\_scores} \leftarrow \texttt{gather}(\texttt{corrector\_logits}, \hat{\mathbf{y}}^{1:L})$
for $\ell = 1$ **to** L **do**
  if $\hat{\mathbf{y}}^{\ell} == \mathbf{y}_t^{\ell}$ **then**
    `# Only consider positions where corrector output is different from input`
    $\texttt{confidence\_scores}^{\ell} \leftarrow -\infty$
  end if
end for
`# 'top_k' is an API that returns the top k argument positions for its first input`
$\texttt{top\_k\_indices} \leftarrow \texttt{top\_k}(\texttt{confidence\_scores}, k)$
for $\ell = 1$ **to** L **do**
  if $\ell \in \texttt{top\_k\_indices}$ **then**
    $\mathbf{y}_t^{\ell} \leftarrow \hat{\mathbf{y}}^{\ell}$
  else
    $\mathbf{y}_t^{\ell} \leftarrow \mathbf{y}_t^{\ell}$
  end if
end for
Return $\mathbf{y}_t^{1:L}$

Sampling Hyperparameters For *ProSeCo*, we match the inference budget of baseline results by using number of unmasking steps equal to $T/4$ per column, performing a corrector loop at every iteration, $\omega = 1$, and applying 3 corrector steps per loop, $S = 3$. We use $k = 100$ as the top-k parameter for Algorithm 4. Evaluation was performed with an EMA checkpoint. During training we used an EMA decay factor of 0.9999.

Baselines Values for the baseline models were taken from Wang et al. [57], except for PRISM results which were taken from Kim et al. [25]. For PRISM, results correspond to the 'PRISM-loop' method presented in Table 3 of Kim et al. [25].

For GIDD [56], we use the $p_u = 0.2$ open-source checkpoint provided by the authors. Since this model was trained on sequence length of $L = 512$, in contrast to the other comparisons and our model which use $L = 1024$, we evaluate the GIDD model with $L = 512$. We use temperature parameter equal to 0.1, and we allow an equal number of unmasking and self-correction steps. Since the self-correction algorithm proposed in GIDD can terminate prior to exhausting the full budget, in Table 4 we denote the GIDD configurations used to match the GIDD results to the various T budgets in Table 2. Note that for certain T, we err on the side of allowing the GIDD model to exceed the total budget T.

Table 4: Parameters used for GIDD [56] unmasking and self-correction budgets corresponding to the results in Table 2.

$T =$	Unmasking steps	Maximum Self Correction Steps	Avg. NFEs
128	64	64	123.9
256	128	128	212.6
512	512	512	601.3
1024	1024	1024	1114.2

D Additional Experimental Results

D.1 Quality-Efficiency Tradeoffs

In Table 5, we compare the baseline method to the *ProSeCo* Max, Balanced, and Fast configurations and report throughput (in tokens per second, TPS) as well as the accuracy and efficiency gains. In Figure 6, we also repeat our Pareto frontier results from Figure 4, with throughput reported.

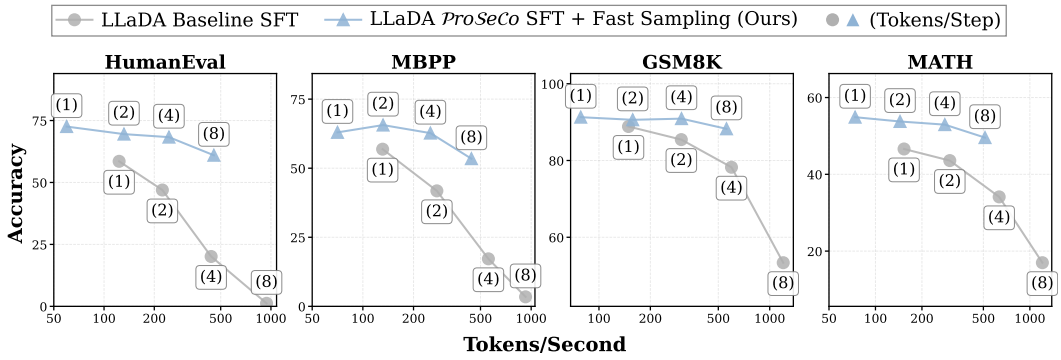

Figure 6: Pareto frontier analysis with throughput (tokens per second, TPS) reported.

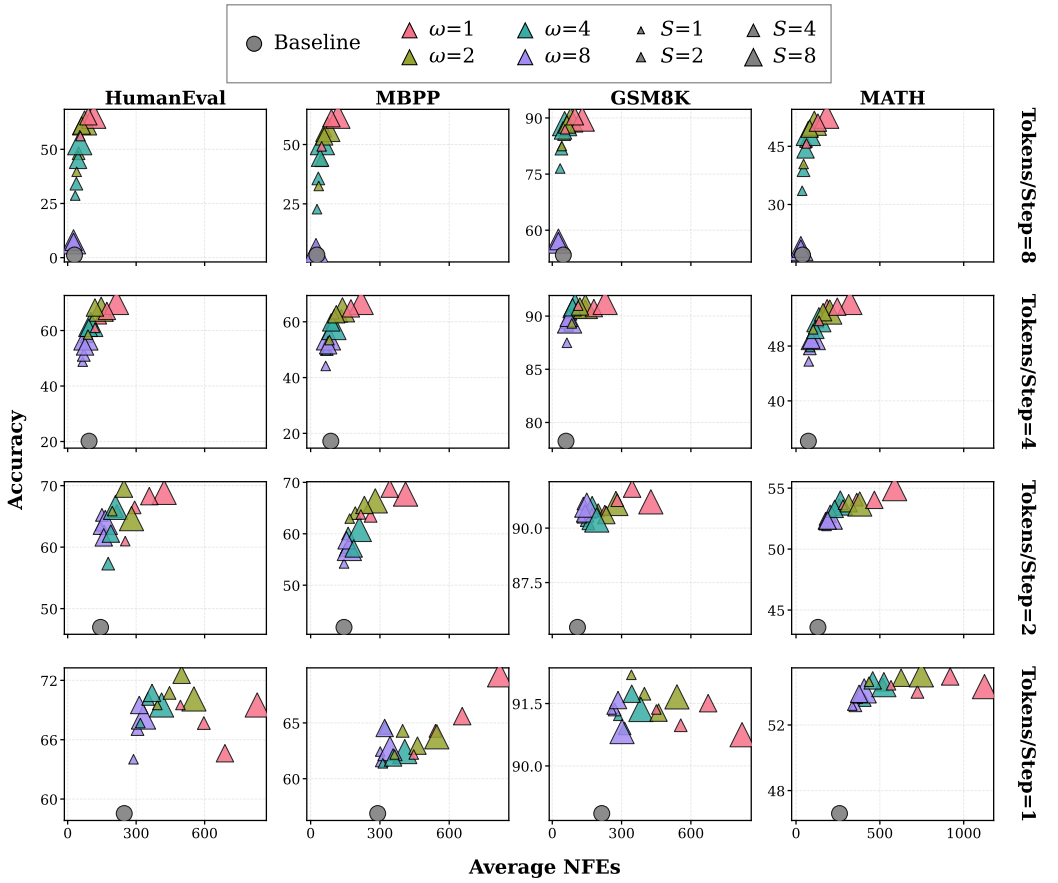

Figure 7: Ablation: Performance across various configurations of corrector steps. Frequency $\omega \in \{1, 2, 4, 8\}$ (denoted by color) and $S \in \{1, 2, 4, 8\}$ (denoted by marker size).

Table 5: Quality vs. Efficiency Metrics. We report Accuracy (%), Average Number of Function Evaluations (NFEs), and Throughput (Tokens/Second). Best accuracy gain and efficiency results are **bolded**. Improvements over the baseline are highlighted in green.

	HumanEval	MBPP	GSM8K	MATH
Baseline (Vanilla SFT)				
Accuracy	58.54	56.88	88.86	46.60
Avg. NFEs	247.8	290.3	214.2	259.4
Tokens/Second	123.14	131.04	148.92	153.15
ProSeCo Max				
Accuracy (+ gain)	**72.56** (+14.0)	**69.31** (+12.4)	**92.19** (+3.3)	**55.06** (+8.5)
Avg. NFEs (× speedup)	499.2 (0.50×)	818.3 (0.35×)	342.5 (0.63×)	747.5 (0.35×)
Tokens/Second (× speedup)	59.66 (0.48×)	37.48 (0.29×)	103.83 (0.70×)	61.38 (0.40×)
ProSeCo Balanced				
Accuracy (+ gain)	70.12 (+11.6)	66.93 (+10.1)	91.28 (+2.4)	54.36 (+7.8)
Avg. NFEs (× speedup)	213.9 (1.16×)	218.7 (1.33×)	228.7 (0.94×)	321.2 (0.81×)
Tokens/Second (× speedup)	137.07 (1.11×)	131.15 (1.00×)	158.33 (1.06×)	141.94 (0.93×)
ProSeCo Fast				
Accuracy (+ gain)	60.98 (+2.4)	62.70 (+5.8)	90.90 (+2.0)	49.56 (+3.0)
Avg. NFEs (× speedup)	**61.4** (4.04×)	**110.9** (2.62×)	**119.5** (1.79×)	**81.2** (3.19×)
Tokens/Second (× speedup)	**455.01** (3.70×)	**251.92** (1.92×)	**304.33** (2.04×)	**512.27** (3.34×)

D.2 Ablation: Robustness to Corrector Parameters

In Figure 7, we demonstrate that our model, which underwent SFT with *ProSeCo* training, is highly robust to the choice of the corrector sampling parameters, i.e., the frequency of corrector loops ω and number of corrector steps per loop S. We look at various levels of unmasking parallelism, ranging from fully sequential at tokens/step $= 1$ to tokens/step $= 8$, and evaluate our model with $\omega \in \{1, 2, 4, 8\}$ and $S \in \{1, 2, 4, 8\}$. In particular, we see that at every level of decoding parallelism, our model can beat the best performing Baseline SFT results for all choices of ω and S.

A few additional observations from Figure 7, in the first two rows, corresponding to 'fast sampling' regimes, in order to match or exceed the best baseline accuracy, which is attained with tokens/step $=$ 1, we require more frequent corrector loops $\omega \in \{1, 2\}$. In these settings we can drastically improve efficiency relative to the baseline model. In this fast sampling regime, we also see a general trend that, for a fixed corrector budget (i.e., $S \cdot L/\omega$) , more frequent but shorter correction loops are typically more effective. Additionally, when unmasking sampling increases to tokens/ step $\in \{1, 2\}$, we find the expected trend that scaling both frequency of correction loops and number of steps per loop leads almost uniformly leads to improved sample quality, at the cost of additional NFEs.

D.3 Selecting the Max, Balanced, and Fast Points

From the hyperparameter sweep described in Appendix D.2, we selected representative points as *ProSeCo* Max, Balanced, and Fast. The chosen parameters for each of these are detailed in Table 3. For the Max configuration, we selected the best overall *ProSeCo* performance from the sweep performed in Figure 7. For the Fast configuration, we selected points from parallel generation of either tokens/step $= 4$ or tokens/step $= 8$ where we attain better accuracy than the best baseline performance, which uses tokens/step $= 1$. For the Balanced configuration, we anchor tokens/step $=$ 4, and select a configuration that pushes the boundary to the north-west across all benchmarks.

D.4 Ablation: Self-correction Sampling with Standard MDM

In Table 6, we demonstrate the importance of *ProSeCo* training, by comparing self-correction sampling applied to a model with standard MDM SFT vs. one with *ProSeCo* loss and Algorithm 1. These results underscore the importance of training a model to self-correct. With standard MDM training, the model does not learn to produce meaningful predictions at already unmasked token locations. Our method unlocks this ability, and therefore benefits from the self-correction sampling algorithm.

Table 6: Ablation: Applying self-correction sampling to models trained with standard MDM loss leads to worse performance, underscoring the importance of the *ProSeCo* framework. Pass@1 accuracy (%) performance is reported.

Training + Sampling	HumanEval	MBPP	GSM8K	MATH
Vanilla SFT + Standard MDM	58.54	56.88	88.86	46.60
Vanilla SFT + *ProSeCo* Balanced	1.83	1.85	60.42	27.06
ProSeCo SFT + *ProSeCo* Balanced	72.56	69.31	92.19	55.06

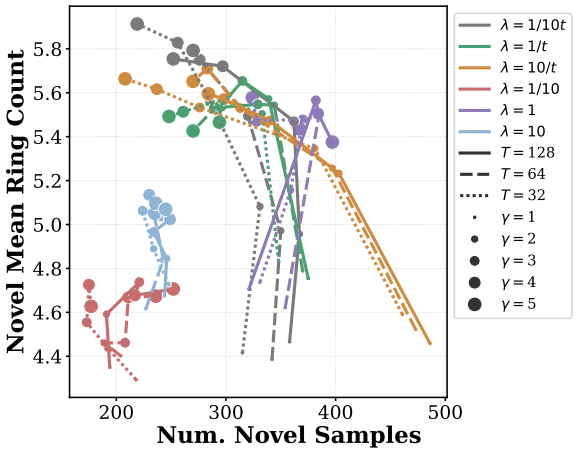

Figure 8: Ablation: Selecting self-correction loss weight λ_t. We retrain the discrete classifier model for maximizing ring count with various λ_t values.

D.5 Ablation: Selecting λ_t

Below we present the ablation on the self-correction loss term weight. We run training for discrete classifier guidance on the ring count property with fixed $\lambda \in \{0.1, 1, 10\}$ and a time-varying weighted $\lambda_t \in \{0.1 \cdot \frac{\dot{\alpha}_t}{(1-\alpha_t)}, 1 \cdot \frac{\dot{\alpha}_t}{(1-\alpha_t)}, 10 \cdot \frac{\dot{\alpha}_t}{(1-\alpha_t)}\}$. We report results in Figure 8. We find that using the $\frac{\dot{\alpha}_t}{1-\alpha_t}$ weighting consistently improves results. Furthermore, our model is robust to different scalings of this time-dependent weighting.

D.6 Error Bars for Table 2

In Table 7, we report mean $\pm$ standard error from four random seeds used during sampling for our model's results on unconditional generation (Table 2).

E Generated Samples

E.1 LLaDA *ProSeCo* SFT Samples

In Figures 9 and 10, we present sample generations for the HumanEval and GSM8K datasets, respectively, using the maximum accuracy configuration for each benchmark (see Table 3).

E.2 *ProSeCo* Unconditional Generation Samples

In Figure 11, we present a sample generated from the *ProSeCo* model trained on OWT. We use total sample budget of $T = 256$, which consists of 64 unmasking steps, a corrector loop every $\omega = 1$ step, and $S = 3$ corrector steps per loop.

Table 7: Mean $\pm$ standard error for OWT unconditional generation metrics for *ProSeCo*.

$T =$	MAUVE (↑)	Gen. PPL (↓)	Entropy (↑)
128	0.311 ± 0.006	23.045 ± 0.031	5.452 ± 0.002
256	0.572 ± 0.006	16.52 ± 0.014	5.376 ± 0.005
512	0.622 ± 0.005	13.196 ± 0.031	5.293 ± 0.001
1024	0.592 ± 0.006	10.92 ± 0.014	5.221 ± 0.005

F Assets

In Table 8, we list the corresponding licenses for datasets used in this work.

Table 8: Datasets and corresponding licenses.

Dataset	Licence
GSM8K [12]	MIT
HumanEval [10]	MIT
Llama-Nemotron: Efficient Reasoning Models [5]	CC BY 4.0
MBPP [3]	MIT
MinveraMath [21]	MIT
OpenWebText [18]	Creative Commons CC0 license (“no rights reserved”)
QM9 [44, 45]	N/A

In Table 9, we list the corresponding licenses for software packages used in this work.

Table 9: Software and corresponding licenses.

Library	License
HuggingFace [62]	Apache 2.0
Hydra [64]	MIT
Fast-DLLM [63]	Apache 2.0
Language Model Evaluation Harness [16]	MIT
Matplotlib [24]	Matplotib license
Mauve [42]	GNU General Public License, Version 3
MDLM [46]	Apache 2.0
Nemo Skills	Apache 2.0
NumPy [20]	NumPy license
OmegaConf	BSD 3-Clause
Pandas [38]	BSD 3-Clause “New" or “Revised"
PyTorch [39]	BSD-3 Clause
PyTorch Lightning [15]	Apache 2.0
RDKit [29]	BSD 3-Clause “New" or “Revised"
Seaborn [60]	BSD 3-Clause “New" or “Revised"
TorchMetrics	Apache 2.0
UDLM [49]	Apache 2.0

G Impact Statement

Our paper is intended as a contribution to the machine learning community. Our work holds the promise for positive impact by accelerating inference and improving quality of language models. While, our method is also subject to the dangers of misuse present in language modeling, especially in the application to biological sequences, given the limited scale of models that we explore, the experiments described in this work do not pose any significant risks.

H LLM Usage Disclosure

Large Language Models (LLMs) were used to format and edit this manuscript and for formatting figures. Additionally, integrated LLMs in the code editing software used while developing this work were used for auto-complete suggestions and in debugging.

LLMs were **not** used for the core development of our methodology, derivation of key results, nor the programming of our main code and experiments.

HumanEval Sample

Prompt shown to the model

````
Here is a problem for which you need to generate/complete code:

def make_a_pile(n):
        """
        Given a positive integer n, you have to make a pile of n levels of
        stones.
        The first level has n stones.
        The number of stones in the next level is:
                - the next odd number if n is odd.
                - the next even number if n is even.
        Return the number of stones in each level in a list, where element at
        index
        i represents the number of stones in the level (i+1).

        Examples:
        >>> make_a_pile(3)
        [3, 5, 7]
        """

  Please continue to complete the function with python programming language.
  You are not allowed to modify the given code and do the completion only.

  The solution should be in the following format:
  ```python
  # Your code here
  ```
````

Model output

````
```python
def make_a_pile(n):
    """
    Given a positive integer n, you have to make a pile of n levels of stones.
    The first level has n stones.
    The number of stones in the next level is:
        - the next odd number if n is odd.
        - the next even number if n is even.
    Return the number of stones in each level in a list, where element at index
    i represents the number of stones in the level (i+1).

    Examples:
    >>> make_a_pile(3)
    [3, 5, 7]
    """
    pile = []
    for i in range(1, n + 1):
        if i % 2 == 1:          # odd level -> next odd number
            pile.append(2 * i - 1)
        else:                   # even level -> next even number
            pile.append(2 * i)
    return pile
```
````

Figure 9: Example of *ProSeCo*'s generated response on the HumanEval dataset using the max accuracy configuration (see Table 3).

GSM8K Sample; Expected answer: 6

Prompt shown to the model

```
Solve the following math problem. Show your step-by-step reasoning, then
provide the final answer inside \boxed{}.

  Solo has to read 4 pages from his Science textbook, 20 pages from his Social
  Studies textbook, 7 pages from his History textbook and 8 pages from his
  Geography textbook. Solo read 15 pages on Monday. If he has 4  more days to
  complete all his reading, how many pages does he need to read, on average, in
  one day?
```

Model output

```
**Step-by-step reasoning**

1. **Total pages to be read**
   \[
   4\ (\text{Science}) + 20\ (\text{Social Studies}) + 7\ (\text{History}) + 8\
   (\text{Geography}) = 39\ \text{pages}
   \]

2. **Pages already read**
   \[
   15\ \text{pages on Monday}
   \]

3. **Pages left to read**
   \[
   39 - 15 = 24\ \text{pages}
   \]

4. **Number of days left**
   \[
   4\ \text{more days}
   \]

5. **Average pages per day**
   \[
   \frac{24\ \text{pages}}{4\ \text{days}} = 6\ \text{pages per day}
   \]

\[
\boxed{6}
\]
```

Figure 10: Example of *ProSeCo*'s generated response on the GSM8K dataset using the max accuracy configuration (see Table 3).

OWT Generated Sample

"|**endoftext**| campaigns trying to figure out how to raise money for a ballot measure America's political dailies have not adopted an information-delivery strategy. Teich's most recent foray was, according to its spokeswoman, a brand new social media campaign that it took a break from advertising the previous year, and launched on its website. Both stem from a tradition of public-sector philanthropy, and position the company as a public utility whose funding is in no way tied to government spending or health care. We have reached out to Teich for more information, and so far have received no response.

But Teich has all but abandoned its information delivery strategy—or at least when it comes to what it calls its "frontiers."

While its Indiegogo campaign has been focused on social issues, much of it has centered on public health and safety initiatives in an effort to reduce the number of traffic deaths, make driving less dangerous, reduce dependence on oil and to improve the health of people in a developing country. Its

budgeted nearly $60 million in fiscal year 2016, according to Teich Institute, a nonprofit that tracks Teich's spending.

Back in 2014, the year it launched, the company calculated how much it could spend on new software, or new nurses, or programs that focus on technology. It spends a big chunk of that money on budgeting—but there are different approaches to health care spending. For instance, some decide what kind of money they spend; others decide how much money they spend on a particular service. So how much Teich spends depends on the investment.

You've heard of this about health-care. The Obama health-care bill seemed to cost $200 billion—but neither the White House nor the administration said anything about how much it cost. Teich has put its total at $20 billion.

"The big question for me is not just whether Teich decides to spend too much, but whether TARP looks at spending too much," says Erin McAllister, executive director of the Teich Public Policy Institute. That calculation helps explain how much Teich spent to fight the opioid epidemic here in Washington D.C., she says, though she says she doesn't know just how much that money went to a particular cause.

Students, for instance, know how to gamble. "They probably knew what they were going to spend," she says. "What surprises me is that people don't know when they buy something that is going to have a bigger impact on an institution than an investment."

Subscribe to Smart Money's Smart Money app—you can download it here.

In some cases, Teich says it didn't know how much of its money was spent, so it didn't disclose how much the company spent. That likely happened in 2010 and 2012, for "from an investment perspective, you're basically going the other way to buy something that's going to have a bigger impact than an investment," McAllister says. "You may see something that has less impact than investment, and you may see something that has more impact and less impact than investment."

In New York City—where Teich's Indiegogo campaign spent nearly $50 million in fiscal year 2016—it managed to convince advocacy groups to spend the bulk of its money on television ads. The company and its partners estimated that the campaign would finance net $2 million in television ads over the next 12 months. In other cities, similar things happen year after year, McAllister says.

So much depends on the investment. The public-sector infrastructure bought by Teich remains relatively small. Its advertising strategies have changed little over the years, and they continue to attract big-name attention.

Of its most recent investments in public health, Ken Farrell, a lawyer with the President's Office, agrees that Teich has designed previous advertising campaigns that have largely paid off. But McAllister says the payoff could be quite big, depending on how much the company spends.

The company's sticker price has been criticized by elected officials, environmental groups and those who ignore the fossil fuel industry at their peril. But its spending on public health initiatives has been relatively steady—it's spent almost $650 million since 2010. Neither Trump or his administration has really pushed for new regulations despite concerns about their potential benefits. And experts aren't totally convinced that new taxes will improve public health—especially because of the already unacceptably high tolls and highway tolls in the United States. Tax advocates will argue that new taxes will be good for everyone—but are skeptical.

"It's compelling," Stan Diem, a political science professor at Wesleyan University says. "But there is no evidence that it is|**endoftext**|"

Figure 11: Generated sample from *ProSeCo* trained on OWT, with a total budget of $T = 256$: 64 unmasking steps, corrector frequency $\omega = 1$ and $S = 3$ steps per loop.