RANDOM REMASKING SCALES MASKED DIFFUSION INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Masked diffusion models (MDMs) generate sequences by progressively unmasking tokens, but errors introduced during decoding are typically irreversible. Recent work has shown that iterative correction can improve sample quality, yet existing theory fails to explain this given a learned, imperfect denoiser model. We reformulate MDM inference as a Markov chain on the clean sequence space, defined by a remask-denoise kernel that randomly re-introduces masks and then denoises. This helps us analyze the actual denoising trajectory, and we examine conditions under which each iteration step contracts the KL divergence. The analysis motivates Random Remask-Denoise ($\mathrm{R}^2\mathrm{D}$) inference, a purely test-time, finetuning-free, reward-free procedure. To further improve performance, we propose Trajectory Correction Training (TCT), a denoiser-training procedure that exposes the model to its own errors along the generation trajectory. On logical tasks, our approach exhibits a striking near-log-linear test-time scaling, achieving over 97% accuracy on hard Sudoku puzzles and over 99% accuracy on 3-SAT. Our approach also applies to pretrained models on actual language tasks, beating iterative correction baselines and the official Q-mode accuracy of LLaDA2.1-mini on HumanEval+.

1 INTRODUCTION

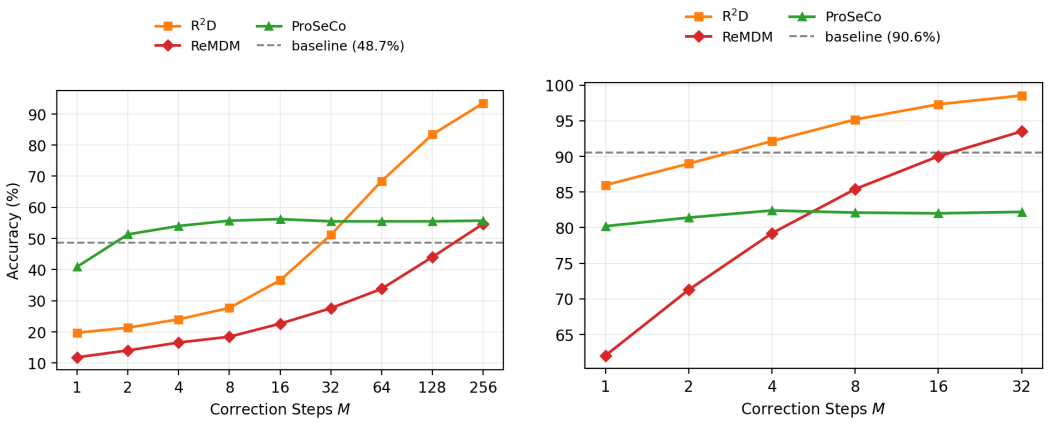
Masked diffusion models (MDMs) have emerged as a strong alternative to autoregressive models (ARMs) for sequence generation (Sahoo et al., 2024; Lou et al., 2024; Ou et al., 2024; Shi et al., 2024). By denoising under bidirectional attention rather than committing to a left-to-right factorization, MDMs can in principle leverage context from both sides when filling in tokens, an advantage that has shown promise on reasoning-heavy tasks.

A central limitation of standard MDM decoding is that revealed tokens are frozen for the rest of the trajectory: Errors introduced by parallel decoding cannot be undone. Iterative correction methods have been proposed to resolve this, either by remasking committed tokens for re-generation (Wang et al., 2026), or by training a model with self-correction ability (Schiff et al., 2026). These methods not only serve as a remedy for parallel-decoding errors, but also unlock *test-time scaling* for MDMs: The number of model evaluations is no longer bounded by the response length. Each correction step can be viewed as an additional unit of compute spent refining the answer, in spirit analogous to chain-of-thought reasoning for autoregressive models, with iterative error correction itself acting as a form of reasoning.

Despite the empirical success, iterative-correction methods fail to explain why they improve generation quality from the theoretical perspective. Existing explanations, e.g., the ReMDM (Wang et al., 2026) and the predictor–corrector framework (Campbell et al., 2022) only provide convergence analysis of the intermediate sequence marginal distribution under a pefect denoising model. They fail to analyze the actual trajectory generated by an imperfect denoiser, which is the question that really matters.

The prevailing formulation treats MDM inference as an iterative denoising process from a fully masked seuqence into a clean sequence. This is hard for a generation trajectory analysis, as intermediate sequences correspond to different masking ratios and hence lie in different subspaces. Observing that each inference step can be further decomposed into a *denoise* step and a *remask* step, we re-formualte the entire MDM inference process in this work as an iteration over the *clean sequence space*: At each step, some of the tokens in the clean sequence are remasked, and then

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



(a) Sudoku-Hard (b) 3-SAT

Figure 1: Inference scaling on Sudoku-Hard and 3-SAT. $\mathrm{R}^2\mathrm{D}$ and ReMDM are run on standard training checkpoints, both using the standard variant; ProSeCo has its own training method. $\mathrm{R}^2\mathrm{D}$ scales near-log-linearly on both tasks and stays above ReMDM at every budget, whereas ProSeCo plateaus after a handful of correction steps.

denoised back into a clean sequence. We identify the *remask–denoise kernel* as the single object underlying both standard decoding and correction, and we characterize the conditions under which each kernel step pulls the chain closer to the data distribution, i.e., the contraction property.

The contraction strength is governed by three factors: Task structure, inference timestep schedule and model quality. Tasks with weaker conditional coupling between tokens scale more steeply with random-remasking compute, e.g., Sudoku and 3-SAT. The timestep schedule should be non-increasing, reducing gradually from 1 (fully masked sequence) to 0 (clean sequence). The challenge is how to decide the number of iterations at each step. We propose a natural solution: Keep iterating at the given step until the sequence converges by observing the clean sequence, which leads to our **Random Remask-Denoise** ($\mathrm{R}^2\mathrm{D}$) inference. The model should be trained under the actual inference trajectory, and thus we propose **Trajectory Correction Training** (TCT), a self-correction training method that is paired with $\mathrm{R}^2\mathrm{D}$ inference.

On logical tasks like Sudoku-Hard and 3-SAT, $\mathrm{R}^2\mathrm{D}$ with standard training shows significant scaling effect (Figure 1), boosting accuracy from 19.7% and 86% to 93.5% and 98.5%. TCT training further increases accuracy to 97.5% and 99.3%, respectively. $\mathrm{R}^2\mathrm{D}$ also works well on language tasks. On the coding benchmark HumanEval+, $\mathrm{R}^2\mathrm{D}$ with off-the-shelf LLaDA2.1-mini surpasses iterative correction baselines and the official Q-mode inference.

Our main contributions are:

- A unified Markov-chain framework for MDM inference, in which standard decoding and self-correction are instances of a single remask–denoise kernel acting on the complete sequence space.
- Random Remask-Denoise ($\mathrm{R}^2\mathrm{D}$), a reward-free, finetuning-free inference algorithm with an adaptive timestep schedule motivated by the contraction bound.
- Trajectory Correction Training (TCT), a denoiser-training procedure that supplies trajectory-aware supervision at every position and pairs naturally with $\mathrm{R}^2\mathrm{D}$.
- Across domains like logical tasks and coding, we demonstrate $\mathrm{R}^2\mathrm{D}$ achieves test-time scaling, and the scaling effect is dependent on the token coupling nature of the task. TCT training further boosts $\mathrm{R}^2\mathrm{D}$ with self-correction ability.

2 RELATED WORK

Iterative-correction inference. A growing body of work equips MDMs with the ability to revise tokens that have already been revealed, falling into two groups. *Training-free* methods (Wang et al., 2026; Shah et al., 2026) modify only the inference procedure of a pretrained denoiser. For instance, ReMDM (Wang et al., 2026) derives a principled remasking sampler from a custom backward process and remasks already-decoded tokens either uniformly at random or by model confidence before re-predicting them. *Trained* approaches instead fine-tune or augment the denoiser to recognize and revise its own mistakes. ProSeCo (Schiff et al., 2026) learns a corrector that consumes the denoiser's own outputs and alternates unmasking with correction at inference. PRISM (Kim et al., 2025), RemeDi (Huang et al., 2025), BackPlay (Liu et al., 2026), and Corrective DLM (Zhang et al., 2026) differ in the training recipe but share the same goal of producing per-token quality signals that drive selective remasking. All of these approaches frame correction defensively, as a way to repair errors introduced by parallel decoding; We instead treat iterative remasking and correction as a reasoning mechanism in its own right and as a primary axis for test-time scaling.

Test-time scaling for reward guidance. Most test-time scaling work for MDMs lies in the reward-guidance regime, in which an external reward function or verifier steers sampling toward higher-quality outputs. These methods scale through three mechanisms: *particle-based sampling* (Singhal et al., 2025; Dang et al., 2026; Ou et al., 2026), *iterative noise–denoise refinement* (Lee et al., 2025; Uehara et al., 2025; Phunyaphibarn & Sung, 2026), and *direct guidance* (Tejaswi et al., 2026; Chen et al., 2025). The closest to our framing is Phunyaphibarn & Sung (2026), which also samples from a Markov chain on the complete sequence space but offers limited theoretical analysis of the sampler's convergence. All three families rely on an external quality signal to guide intermediate generation, whereas the inference algorithms we study are reward-free.

3 PRELIMINARIES

We briefly review the absorbing masked diffusion model (MDM) setup that our method builds on.

Setup. Let $\mathbf{x}_0 = (\mathbf{x}_0^1, \dots, \mathbf{x}_0^L) \in \mathcal{V}^L$ be a clean discrete sequence of length L, where $\mathcal{V}$ is a finite vocabulary. We augment the vocabulary with a special mask token $\mathbf{m}$ and write $\mathbf{x}_t \in (\mathcal{V} \cup \{\mathbf{m}\})^L$ for the corrupted sequence at diffusion time $t \in [0, 1]$. Let $\alpha_t \in [0, 1]$ be a monotonically decreasing noise schedule with $\alpha_0 = 1$ and $\alpha_1 = 0$. Clean sequences are drawn from a data distribution p_{data} over $\mathcal{V}^L$.

Diffusion training objective. An MDM learns a network f_θ whose token-wise output $f_\theta^i(\cdot \mid \mathbf{x}_t) \in \Delta(\mathcal{V})$ predicts the clean token given the current intermediate sequence $\mathbf{x}_t$. Standard MDMs adopt carry-over unmasking, i.e., $f_\theta^i(\mathbf{x}_t^i \mid \mathbf{x}_t) = 1$ if $\mathbf{x}_t^i \in \mathcal{V}$, while self-correction models relax this constraint. The training objective simplifies to a weighted cross-entropy loss over masked tokens (Ou et al., 2024; Sahoo et al., 2024; Shi et al., 2024):

$$\mathcal{L}_{\text{MDM}}(\theta) = \mathbb{E}_{\mathbf{x}_0 \sim p_{\text{data}},\, t,\, \mathbf{x}_t \sim q_{t|0}(\cdot|\mathbf{x}_0)} \left[\frac{\dot{\alpha}_t}{1 - \alpha_t} \sum_{i=1}^{L} \mathbf{1}\{\mathbf{x}_t^i = \mathbf{m}\} \big(-\log f_\theta^i(\mathbf{x}_0^i \mid \mathbf{x}_t) \big) \right].$$

Standard MDM training thus teaches the model to recover each clean token independently at every currently masked position. Because the per-position outputs are conditionally independent given $\mathbf{x}_t$, the network induces a joint distribution

$$f_\theta(\mathbf{x}_0 \mid \mathbf{x}_t) \;:=\; \prod_{i=1}^{L} f_\theta^i(\mathbf{x}_0^i \mid \mathbf{x}_t),$$

which we use in Section 4 as the model's posterior over clean sequences.

Standard diffusion inference. At inference time, generation starts from the fully masked sequence $\mathbf{x}_{t_0} = (\mathbf{m}, \dots, \mathbf{m})$ and proceeds along a decreasing time grid $1 = t_0 > t_1 > \cdots > t_K = 0$. At each step, the model predicts token distributions $f_\theta^i(\cdot \mid \mathbf{x}_{t_j})$ at all masked positions, and an unmasking

policy selects a subset $S_j \subseteq \{i : \mathbf{x}_{t_j}^i = \mathbf{m}\}$ of size $(\alpha_{t_{j+1}} - \alpha_{t_j})L$ to reveal — either uniformly at random or by model confidence. For each $i \in S_j$, a token is drawn from $f_\theta^i(\cdot \mid \mathbf{x}_{t_j})$ (by sampling or arg-max) and written into the sequence to produce $\mathbf{x}_{t_{j+1}}$. Standard inference never revises an already revealed token: Once a position is unmasked, it stays fixed for the remainder of the trajectory. Our work departs from this convention by allowing iterative correction of previously revealed tokens.

4 METHOD

The standard view of MDM inference is an iteration over the intermediate sequence space: Starting at $t = 1$, the noise level decreases monotonically until a clean sequence $\mathbf{x}_0$ is produced. This view is faithful to the underlying diffusion theory but has two limitations. First, correction is treated as an engineering add-on for fixing model errors rather than an inherent part of inference. Second, the framework admits no dynamic analysis of an imperfect model: Because the iteration variable $\mathbf{x}_t$ changes type at every step, there is no unified metric for tracking how close the current state is to the data distribution.

Observe that each inference step can be further decomposed into a *denoise* step and a *remask* step. In the denoise step, the model predicts the clean sequence $\mathbf{x}_0$ from any intermediate sequence $\mathbf{x}_t$, and in the remask step, some of the tokens in $\mathbf{x}_0$ are turned back into masks, resulting in another intermediate sequence at another noise level $\mathbf{x}_{t'}$. In other words, the MDM inference process, either standard denoising or iterative correction, can be unified into the following transition:

$$\begin{aligned}&\mathbf{x}_{t_0} \to \mathbf{x}_0 \to \mathbf{x}_{t_1} \to \mathbf{x}_0 \to \cdots \to \mathbf{x}_{t_n}\\ &t_0 = 1, t_n = 0\end{aligned}$$

Therefore, we can view the inference process as an iteration over the clean sequence space $\mathcal{V}^L$. Under a Markovian remask step, the transition is given by a *remask–denoise kernel*:

$$\tilde{K}_t^\theta(\mathbf{x}_0' \mid \mathbf{x}_0) \;=\; \sum_{\mathbf{x}_t} f_\theta(\mathbf{x}_0' \mid \mathbf{x}_t)\, q_{t|0}(\mathbf{x}_t \mid \mathbf{x}_0),$$

which approximates the Bayes-optimal counterpart

$$K_t(\mathbf{x}_0' \mid \mathbf{x}_0) \;=\; \sum_{\mathbf{x}_t} q_{0|t}(\mathbf{x}_0' \mid \mathbf{x}_t)\, q_{t|0}(\mathbf{x}_t \mid \mathbf{x}_0).$$

We refer to $\tilde{K}_t^\theta$ as the *model kernel* and to K_t as the *true kernel*. The denoiser in the kernel uses the trained model's raw distribution f_θ. In practice, people often use an alternative sampler $h\colon \Delta(\mathcal{V}) \to \Delta(\mathcal{V})$ to adjust the model prediction, e.g. $\arg\max$ sampler. In this case, the analysis still holds by replacing $h \circ f_\theta$ in place of f_θ. Our re-formulation follows the predictor-corrector framework (Campbell et al., 2022), indicating that K_t converges to the ground truth distribution p_{data}.

Proposition 1 (Stationarity of the true kernel, proof in Section A.1.)**.** *For every $t \in (0, 1)$, the kernel K_t admits p_{data} as a stationary distribution: $p_{\text{data}} K_t = p_{\text{data}}$.*

Proposition 1 implies that the model kernel $\tilde{K}_t^\theta$ must be close to the true kernel K_t for the chain to converge near p_{data}, which in turn requires the remasking step to be strictly Markovian. However, the remask step in standard diffusion inference violates this requirement, since it only remasks a previous mask, which is history information inaccessible in $\mathbf{x}_0$. In our formulation non-Markovian remasking is an unnecessary source of error, and thus we recommend remasking every token in the clean sequence $\mathbf{x}_0$ uniformly at random, ensuring the Markov property.

A more important yet unanswered question is: **Under what conditions does the actual Markov chain induced by $\tilde{K}_t^\theta$ converge?** To analyze it, we fix any $p \in \Delta(\mathcal{V}^L)$ on the chain's state space and write $p' = p\tilde{K}_t^\theta$ for its image after one kernel step, where $(p\tilde{K}_t^\theta)(\mathbf{x}_0') := \sum_{\mathbf{x}_0} p(\mathbf{x}_0)\, \tilde{K}_t^\theta(\mathbf{x}_0' \mid \mathbf{x}_0)$. Convergence is measured by $\mathrm{KL}(p_{\text{data}} \| p)$, and a sufficient condition is the *contraction property*: $\mathrm{KL}(p_{\text{data}} \| p') < \mathrm{KL}(p_{\text{data}} \| p)$ at every p that arises along the inference trajectory.

Proposition 2 (KL contraction bound for one correction step)**.** *For any distribution p over $\mathcal{V}^L$ and kernel step t, let $\tilde{q}_t(\mathbf{x}_t) = \sum_{\mathbf{x}_0} q_{t|0}(\mathbf{x}_t \mid \mathbf{x}_0)\, p(\mathbf{x}_0)$ be the marginal obtained by remasking p, and*

define the contraction ratio *of the masking channel*

$$\rho_t(p) \;=\; \frac{\mathrm{KL}(q_t \,\|\, \tilde{q}_t)}{\mathrm{KL}(p_{\mathrm{data}} \,\|\, p)}, \tag{1}$$

which satisfies $\rho_t(p) \in [0,1]$ *by the data-processing inequality. Then*

$$\mathrm{KL}(p_{\mathrm{data}} \,\|\, p') \;\le\; \rho_t(p) \cdot \mathrm{KL}(p_{\mathrm{data}} \,\|\, p) \;+\; \epsilon_t,$$

where

$$\epsilon_t \;=\; \mathbb{E}_{\mathbf{x}_t \sim q_t}\big[\mathrm{KL}\big(q_{0|t}(\cdot \mid \mathbf{x}_t) \,\big\|\, f_\theta(\cdot \mid \mathbf{x}_t)\big)\big].$$

The proof, which derives the stronger equality form, is in Section A.2. The two terms in Proposition 2 have distinct interpretations:

- $\rho_t(p)$, the *contraction coefficient*, is monotonically decreasing in t: A larger t destroys more information about $\mathbf{x}_0$, making the two marginals more similar (data-processing inequality), so the term shrinks smoothly from 1 to 0 as t ranges from 0 to 1.
- ϵ_t is the *denoising error*: The expected KL divergence between the Bayes-optimal posterior $q_{0|t}(\cdot \mid \mathbf{x}_t)$ and the model's $f_\theta(\cdot \mid \mathbf{x}_t)$. At large t the input is mostly masked and prediction is harder, so ϵ_t typically increases with t.

Three factors govern the contraction strength: Task structure, timestep schedule, and model quality. We analyze each in turn in the following subsections.

4.1 Task Structure: When Does Random Remasking Help?

For fixed t and p, the contraction ratio $\rho_t(p)$ is task-specific. The following decomposition expresses it in terms of conditional independence between revealed and masked tokens.

Proposition 3 (Subsampling-visibility expression of $\rho_t(p)$)**.** *Let $S \subseteq [L]$ be the random set of revealed positions induced by the forward process at noise level t (independent masking with retention probability α_t, or uniform remasking with $|S| = \lfloor \alpha_t L \rfloor$), and let $\bar{S} = [L] \setminus S$. Write p^S_{data}, p^S for the marginals of p_{data}, p on positions in S, and $p^{\bar{S}|S}_{\mathrm{data}}, p^{\bar{S}|S}$ for the corresponding conditionals. Then for any p on $\mathcal{V}^L$,*

$$\rho_t(p) \;=\; 1 \;-\; \frac{\mathbb{E}_S\, \mathbb{E}_{\mathbf{x}^S \sim p^S_{\mathrm{data}}}\Big[\mathrm{KL}\Big(p^{\bar{S}|S}_{\mathrm{data}}(\cdot \mid \mathbf{x}^S) \,\Big\|\, p^{\bar{S}|S}(\cdot \mid \mathbf{x}^S)\Big)\Big]}{\mathrm{KL}(p_{\mathrm{data}} \,\|\, p)}.$$

The conditional KL term in the numerator captures how much information the revealed positions $\mathbf{x}^S$ leave undetermined about the masked positions $\bar{S}$: The more independent the tokens, the larger this term, the smaller $\rho_t(p)$, and the faster the contraction. Sudoku and 3-SAT are two good examples that have sparse, local constraint structure—each Sudoku cell interacts only with its row, column, and 3×3 box, and each 3-SAT variable appears in only a few clauses—so once a moderate fraction of positions is revealed, each masked token is pinned down by its local neighborhood rather than by long-range interactions. In short, tasks with weakly correlated tokens, or with errors localized to a few positions, benefit more from random-remasking scaling.

4.2 Schedule: Random Remask-Denoise ($\mathrm{R}^2\mathrm{D}$) Inference

The next question is how to choose the timestep t at each step of the chain. The two terms of Proposition 2 tug in opposite directions: A non-increasing schedule is intuitively right, since early on $\mathrm{KL}(p_{\mathrm{data}} \| p)$ is large and a larger t buys faster contraction even at the cost of higher denoising error ϵ_t, whereas late in the chain $\mathrm{KL}(p_{\mathrm{data}} \| p)$ is small and only a smaller t keeps ϵ_t from dominating. ProSeCo (Schiff et al., 2026) skips remasking entirely during correction (i.e. $t = 0$), which is suboptimal under Proposition 2 since $\rho_0(p) = 1$ and the bound provides no contraction. Standard inference and ReMDM (Wang et al., 2026) follow this intuition, but the latter uses a fixed uniform schedule by default, which is suboptimal: Early small-t steps are wasted while $\rho_t(p)\,\mathrm{KL}(p_{\mathrm{data}} \| p)$ still dominates, and late large-t steps are bottlenecked by ϵ_t and yield little further contraction. At any given stage there is an effective operating point $t^\star$ that balances the two terms.

Algorithm 1 Random Remask-Denoise ($\mathrm{R}^2\mathrm{D}$) Inference (standard & correction variants).

Require: Trained MDM f_θ, decoding steps K, per-level correction budget M, stability threshold τ

1: $\mathbf{x}_1 \leftarrow \mathbf{m}^L$ ▷ initialize fully masked, $t = 1$
2: **for** $i = K-1, K-2, \cdots, 0$ **do**
3: $t \leftarrow i/K$
4: $\hat{\mathbf{x}}_0^{\text{prev}} \leftarrow \texttt{None}$; $c \leftarrow 0$ ▷ consecutive-repetition counter
5: **for** $j = 1, 2, \cdots, M+1$ **do** ▷ repeat denoise-remask at noise level t
6: $\hat{\mathbf{x}}_0 \leftarrow \arg\max f_\theta(\cdot \mid \mathbf{x}_t)$
7: **if standard** variant **then** ▷ skipped by the *correction* variant
8: $(\hat{\mathbf{x}}_0)_\ell \leftarrow (\mathbf{x}_t)_\ell$ for ℓ with $(\mathbf{x}_t)_\ell \neq \mathbf{m}$ ▷ freeze revealed tokens
9: **end if**
10: $c \leftarrow c+1$ **if** $\hat{\mathbf{x}}_0 = \hat{\mathbf{x}}_0^{\text{prev}}$ **else** 0
11: $\hat{\mathbf{x}}_0^{\text{prev}} \leftarrow \hat{\mathbf{x}}_0$
12: $n \leftarrow \lfloor (1-t)\, L \rfloor$ ▷ # positions to reveal at noise level t
13: sample $S \subseteq \{1, \cdots, L\}$ uniformly with $|S| = n$
14: $(\mathbf{x}_t)_\ell \leftarrow (\hat{\mathbf{x}}_0)_\ell$ for $\ell \in S$; $(\mathbf{x}_t)_\ell \leftarrow \mathbf{m}$ for $\ell \notin S$ ▷ remask
15: **if** $c \geq \tau$ **then** ▷ converged: $\tau+1$ identical predictions in a row
16: **break**
17: **end if**
18: **end for**
19: **end for**
20: **return** $\hat{\mathbf{x}}_0$

We therefore propose **Random Remask-Denoise** ($\mathrm{R}^2\mathrm{D}$) inference, given in Algorithm 1, which uses an adaptive timestep schedule. The noise variable is discretized into K equally spaced grid points $t \in \{(K-1)/K, (K-2)/K, \cdots, 1/K, 0\}$, with $K = L$ as a natural default (one outer step per token). Starting from the largest t, we hold the noise level fixed and iterate the model kernel $\tilde{K}_t^\theta$, treating the inner loop as a search for a fixed point of $\tilde{K}_t^\theta$ at that level. Under argmax sampling, convergence can be detected once $\hat{\mathbf{x}}_0$ has remained unchanged across τ consecutive denoise steps. We set a per-level budget M so that inference ends with at most $K(M+1)$ forward passes, but our experiments show that actual inference uses far fewer forward passes.

We instantiate $\mathrm{R}^2\mathrm{D}$ in two variants depending on whether the denoiser has been trained for correction, differing only in Line 7 of Algorithm 1. The *standard* variant executes that line, carrying over the revealed positions of $\mathbf{x}_t$ and filling in only the masked entries, which matches the behavior assumed by an off-the-shelf MDM denoiser. The *correction* variant skips it, overwriting every position with the model's prediction so that previously revealed tokens are revised whenever the denoiser disagrees with them. In either case, $\mathrm{R}^2\mathrm{D}$ is a purely test-time procedure that requires no retraining and applies to any pretrained MDM denoiser, with or without explicit correction capability. Note that the standard variant also has error correction ability through random remasking.

4.3 Model Quality: Trajectory Correction Training (TCT)

Having fixed the schedule, we now turn to the denoiser. Bounding $\mathrm{KL}(p_{\text{data}} \| p')$ directly leads to a trajectory-aware ELBO over the denoiser parameters θ.

Proposition 4 (Trajectory-aware training bound)**.** *For any distribution p over $\mathcal{V}^L$ and timestep t, let $p' = p\tilde{K}_t^\theta$ and $\tilde{q}_t(\mathbf{x}_t) = \sum_{\mathbf{x}_0} q_{t|0}(\mathbf{x}_t \mid \mathbf{x}_0)\, p(\mathbf{x}_0)$. Then*

$$\mathrm{KL}(p_{\text{data}} \,\|\, p') \;\leq\; \mathbb{E}_{\mathbf{x}_t \sim \tilde{q}_t,\, \mathbf{x}_0 \sim p_{\text{data}}}\big[-\log f_\theta(\mathbf{x}_0 \mid \mathbf{x}_t)\big] - H(p_{\text{data}}),$$

where $H(p_{\text{data}}) = -\mathbb{E}_{\mathbf{x}_0 \sim p_{\text{data}}}[\log p_{\text{data}}(\mathbf{x}_0)]$ is the (Shannon) entropy of the data distribution.

Since $H(p_{\text{data}})$ is independent of θ, minimizing the bound reduces to a cross-entropy objective on inputs $\mathbf{x}_t$ drawn from $\tilde{q}_t$ — the marginal of the chain's own state p remasked to level t. The proof is in Section A.4.

Faithfully emulating the inference trajectory of Algorithm 1 during training would spend many gradient steps on a single example before the chain advances. Inspired by progressive-unmasking training (PUMA) (Kim et al., 2026), we make three simplifications:

- Omit the correction inner loop, keeping only the outer decoding steps;
- Reduce the decoding trajectory to a fixed stage budget K;
- Use a low-confidence reveal policy to maintain generation quality under that limited budget.

The resulting procedure is Trajectory Correction Training, whose pseudo-code is given in Algorithm 2 (Section B). It maintains B independent decoding chains in parallel, with chain j at stage $n^{(j)} \in [0, K]$ of an unmasking trajectory targeting the paired clean sample $\mathbf{x}_0^{(j)}$. Unlike PUMA, which trains on the standard MDM loss, our objective is the *correction loss*: Token-level cross-entropy of f_θ against the clean target evaluated at *every* position, masked or already revealed,

$$L_{\text{correct}}(\theta) \;=\; -\frac{1}{B}\sum_{j=1}^{B}\sum_{i=1}^{L}\log f_\theta^i\big(x_0^{(j),i} \,\big|\, \tilde{\mathbf{x}}^{(j)}\big). \tag{2}$$

Including the revealed positions is what makes this a correction objective rather than the usual MDM denoising one: Tokens that the model previously filled in incorrectly contribute a non-trivial loss, training f_θ to override its own past mistakes instead of merely copying its own outputs.

5 Experiments

Our experiments are designed to answer three questions:

- How do $\mathrm{R}^2\mathrm{D}$ and TCT compare against iterative correction baselines? (Sections 5.1 and 5.2)
- Does random remasking enable test-time scaling? (Section 5.3)
- How efficient is $\mathrm{R}^2\mathrm{D}$ inference compared with other iterative correction approaches? (Section 5.4)

Inference approaches. We evaluate $\mathrm{R}^2\mathrm{D}$ in both its standard and correction variants, referred to as "$\mathrm{R}^2\mathrm{D}$ (standard)" and "$\mathrm{R}^2\mathrm{D}$ (correction)", respectively, alongside two baselines: ReMDM (Wang et al., 2026) and ProSeCo (Schiff et al., 2026). Both baselines admit an arbitrarily large compute budget and have been reported to exhibit test-time scaling. ProSeCo inference requires its own training method, while ReMDM works with any existing model. In addition, we report the accuracy of other correction training (RemeDi (Huang et al., 2025), PRISM (Kim et al., 2025), BackPlay (Liu et al., 2026)) and non-correction training (Standard, PUMA (Kim et al., 2026)) variants.

Implementation variants of ReMDM. When evaluating ReMDM with models trained with TCT, we use argmax sampling instead of the official stochastic sampling to keep the best performance. In analogy to $\mathrm{R}^2\mathrm{D}$, we use "ReMDM (correction)" to refer to the argmax sampling variant, and "ReMDM (standard)" for the original stochastic categorical sampling implementation that is used for standard-trained models. We include a detailed comparison between different ReMDM configurations in Section C.4.

5.1 Logical Tasks

We first evaluate on two logical reasoning tasks. One is **Sudoku-Hard**, constructed from the training split of Sudoku-Extreme (Wang et al., 2025), retaining only puzzles with rating > 10, which yields 2.3M instances. The last 1,000 form the test set and the rest form the training set. Sudoku difficulty is measured by *rating*, i.e., the number of backtracks needed by the tdoku solver (Dillion, 2025). These puzzles are challenging even for experienced human solvers and dedicated solver programs, requiring substantial trial-and-error search. The model in both tasks is a 6.8M Qwen2-style decoder-only transformer with hidden size 256 and 8 layers and is trained from scratch.

Table 1 compares the accuracy against prior work on Sudoku-Hard and 3-SAT. The per-level correction budget M is set to 256 for Sudoku-Hard and 32 for 3-SAT. We use a smaller budget for 3-SAT because performance almost saturates at $M = 32$. ReMDM and ProSeCo use the same compute budget for a fair comparison. $\mathrm{R}^2\mathrm{D}$ with standard training already surpasses other baselines on both tasks. TCT training further pushes the limit to 97.5% on Sudoku-Hard and 99.3% on 3-SAT. On the same TCT checkpoint and at the same budget, $\mathrm{R}^2\mathrm{D}$ outperforms ReMDM (correction) by 5.2 and 2.8 points, so the gain comes from the remask–denoise kernel and not from the training alone.

Table 1: Accuracy (in %) on Sudoku-Hard and 3-SAT, organized by training method and inference method. ReMDM, R^2D and ProSeCo use the same compute budget (256 steps per token for Sudoku-Hard, 32 steps per token for 3-SAT). R^2D is the primary driver for inference-time scaling, and TCT cooperates well with R^2D to further raise performance.

Training method	Inference method	Sudoku-Hard	3-SAT
Standard	Standard	48.7	90.6
	ReMDM (standard)	54.7	93.5
	R^2D (standard, $M = 1$)	19.7	86.0
	R^2D (standard)	93.5	98.5
PUMA	Standard	50.8	79.6
RemeDi	RemeDi	50.2	84.3
PRISM	PRISM	52.2	86.0
BackPlay	BackPlay	51.9	87.9
ProSeCo	ProSeCo	55.7	82.2
TCT (ours)	Standard	51.2	79.7
	ReMDM (correction)	92.3	96.5
	R^2D (correction, $M = 1$)	43.7	90.0
	R^2D (correction)	**97.5**	**99.3**

5.2 Coding Benchmark

To examine how R^2D behaves on practical language tasks with off-the-shelf models, we further evaluate on the coding benchmark HumanEval+ (Liu et al., 2023) with LLaDA2.1-mini (Bie et al., 2026), a 16B diffusion language model already trained with self-correction ability. Figure 2 reports the results. Besides the three iterative correction methods we run ourselves, we also quote the model's own published Q-mode and S-mode decoding results. R^2D reaches 85.4%, the top accuracy even beyond the model's published Q-mode accuracy (82.9%), and it surpasses ProSeCo and ReMDM, two iterative correction baselines.

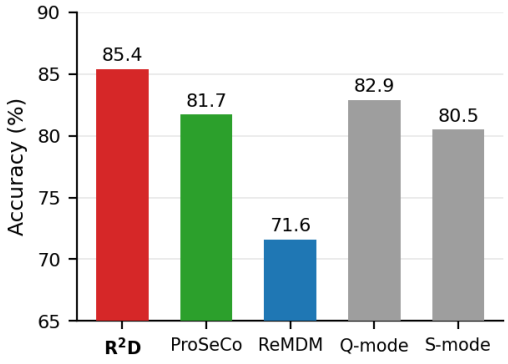


Figure 2: HumanEval+ accuracy with the off-the-shelf LLaDA2.1-mini (16B). The three colored bars are inference methods we run ourselves, and the two grey bars are the model's own published Q-mode and S-mode results. R^2D achieves the top accuracy, surpassing the official Q-mode performance.

5.3 Test-time Scaling

The next question is how test-time performance scales with compute budget, so we compare the accuracy of R^2D, ReMDM and ProSeCo against the per-level correction budget M on logical tasks. R^2D and ReMDM are evaluated with standard training checkpoints, while ProSeCo uses its own training method. As shown in Figure 1, R^2D scales near-log-linearly: From 19.7% at $M = 1$ to 93.5% at $M = 256$ on Sudoku-Hard, and from 86.0% to 98.5% at $M = 32$ on 3-SAT. ReMDM also improves with the budget but stays below R^2D at every budget. ProSeCo plateaus early at low performance: Its correction steps do not remask ($t = 0$), where $\rho_0(p) = 1$ and Proposition 2 guarantees no contraction.

5.4 Compute Comparison

Iterative correction methods buy accuracy with more model forward passes, so a fair comparison should account for compute as well as accuracy. Figure 3 reports the averaged number of function

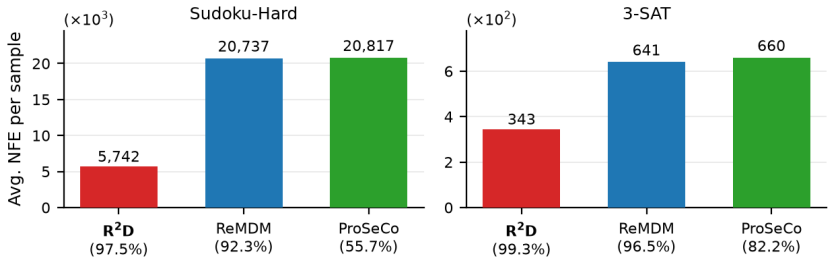


Figure 3: Averaged number of function evaluations (NFE) per sample at the budget of Table 1 (lower is better), with accuracy in parentheses (TCT checkpoint for $\mathrm{R}^2\mathrm{D}$ and ReMDM). $\mathrm{R}^2\mathrm{D}$ exits early, whereas ReMDM and ProSeCo spend their full budget.

evaluations (NFE) per sample that each method spends at the budget of Table 1. $\mathrm{R}^2\mathrm{D}$ and ReMDM use the TCT-trained checkpoint. ReMDM and ProSeCo have no stopping rule and spend their entire budget. $\mathrm{R}^2\mathrm{D}$ applies early exit (Line 15 of Algorithm 1) as soon as its prediction $\hat{\mathbf{x}}_0$ stabilizes, so it spends only about $1/4$ and $1/2$ of ReMDM's compute and reaches higher accuracy.

5.5 Ablation Study

$\mathrm{R}^2\mathrm{D}$ design. We ablate our inference approach on three critical designs: Uniform remasking, early exit, and $\arg\max$ decoding. For uniform remasking, we replace it with low-confidence remasking, a popular strategy used in standard inference. For early exit, we remove it as ablation. For $\arg\max$ decoding (Line 6 of Algorithm 1), we replace it with stochastic sampling from the denoiser's raw distribution. Table 6 (Section D.1) reports the accuracy on Sudoku-Hard for each configuration across different compute budgets. Low-confidence remasking almost removes the scaling effect: Accuracy moves only from 68.7% to 69.6% across a $16\times$ increase in budget, where uniform remasking goes from 65.8% to 97.5% over the same sweep. Removing early exit preserves the scaling trend but pays heavily for it: At $M = 256$ the NFE is nearly 4 times that of the default configuration. Moreover, its accuracy is still lower than that of the default configuration, because random remasking can move the sequence away from the stationary point after convergence. Stochastic sampling loses 10.4 points on accuracy because TCT trains the denoiser along an $\arg\max$ trajectory. It also spends more NFE, as a stochastic prediction makes it harder to fulfill the early-exit test.

TCT design. We ablate TCT on its two designs in Algorithm 2: The top-k unmasking position selection (Line 11), and the $\arg\max$ sampling (Line 9). For position selection, we replace top-k with a random choice among the masked positions; For the reveal operator, we replace $\arg\max$ with temperature-1 sampling. We report the accuracy on Sudoku-Hard under $\mathrm{R}^2\mathrm{D}$ inference, using compute budget $M = 256$, as shown in Table 7 (Section D.2). Position selection is the load-bearing choice: Revealing random positions costs 12.0 points, dropping accuracy from 97.5% to 85.5%, because top-k keeps the training trajectory close to a valid partial solution rather than letting it degrade into a corrupted one. The reveal operator is a minor lever by comparison, costing 0.7 points, consistent with its role of matching the $\arg\max$ decoding that $\mathrm{R}^2\mathrm{D}$ uses at inference.

6 Conclusion

We presented a unified Markov-chain view of masked diffusion inference in which standard decoding and self-correction arise from a single remask–denoise kernel acting on the clean sequence space. The contraction analysis motivates Remask-Denoise ($\mathrm{R}^2\mathrm{D}$) inference and Trajectory Correction Training (TCT), which together produce a nearly log-linear scaling effect on Sudoku-Hard and 3-SAT. Our inference approach also works with pretrained diffusion models on practical language tasks, surpassing published Q-mode accuracy on HumanEval+ with LLaDA2.1-mini. In addition, $\mathrm{R}^2\mathrm{D}$ adopts an early-exit mechanism and is thus more compute-efficient. Applying $\mathrm{R}^2\mathrm{D}$ and TCT to other domains, such as biological sequence design, is a promising future direction.

AI USE STATEMENT

We used generative AI tools as general-purpose coding and writing assistants: for implementation support in the experiment code and for polishing the prose of this manuscript. Research ideas, the theoretical analysis, the experimental design, and all reported results are the authors' own. We have reviewed all AI-assisted work — AI-assisted code was verified and tested by the authors, and all AI-assisted text was checked for accuracy — and we take responsibility for the final content of this work.

In this work, we used generative AI tools for the following:

- Formulating mathematical claims;
- Providing critical ingredients for proving mathematical claims;
- Assisting in the writing of proofs;
- Implementing methods;
- Interpreting results;
- Drafting parts of a research paper;
- Editing a research paper to improve readability.

We have not used generative AI tools for the following:

- Generating synthetic data sets;
- Helping develop theoretical models or conceptual frameworks;
- Proposing or refining hypotheses;
- Assisting with translation;
- Cleaning and reformatting datasets;
- Supporting qualitative and thematic data analysis.

We have manually checked and edited the proofs generated by AI. We have gone through the whole paper and edited manually for the final version of the paper. We have verified the code generated by LLM and the results created.

ETHICS STATEMENT

This work studies inference procedures for masked diffusion models on synthetic reasoning benchmarks (Sudoku-Hard, 3-SAT) and a public code-generation benchmark (HumanEval+). It involves no human subjects, no personally identifiable data, and no new data release. We do not foresee direct harmful applications beyond the general dual-use considerations that apply to improved generative sequence models.

REPRODUCIBILITY STATEMENT

Experiment details are included in Section E. We also release code at `https://anonymous.4open.science/r/r2d-release-E834/README.md`.

REFERENCES

Tiwei Bie, Maosong Cao, Xiang Cao, Bingsen Chen, Fuyuan Chen, Kun Chen, Lun Du, Daozhuo Feng, Haibo Feng, Mingliang Gong, Zhuocheng Gong, Yanmei Gu, Jian Guan, Kaiyuan Guan, Hongliang He, Zenan Huang, Juyong Jiang, Zhonghui Jiang, Zhenzhong Lan, Chengxi Li, Jianguo Li, Zehuan Li, Huabin Liu, Lin Liu, Guoshan Lu, Yuan Lu, Yuxin Ma, Xingyu Mou, Zhenxuan Pan, Kaida Qiu, Yuji Ren, Jianfeng Tan, Yiding Tian, Zian Wang, Lanning Wei, Tao Wu, Yipeng Xing, Wentao Ye, Liangyu Zha, Tianze Zhang, Xiaolu Zhang, Junbo Zhao, Da Zheng, Hao Zhong, Wanli Zhong, Jun Zhou, Junlin Zhou, Liwang Zhu, Muzhi Zhu, and Yihong Zhuang. LLaDA2.1: Speeding Up Text Diffusion via Token Editing, February 2026. URL `http://arxiv.org/abs/2602.08676`. arXiv:2602.08676 [cs].

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, December 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/b5b528767aa35f5b1a60fe0aaeca0563-Abstract-Conference.html.

Tianlang Chen, Minkai Xu, Jure Leskovec, and Stefano Ermon. RFG: Test-Time Scaling for Diffusion Large Language Model Reasoning with Reward-Free Guidance, September 2025. URL http://arxiv.org/abs/2509.25604. arXiv:2509.25604 [cs].

Meihua Dang, Jiaqi Han, Minkai Xu, Kai Xu, Akash Srivastava, and Stefano Ermon. Inference-Time Scaling of Diffusion Language Models via Trajectory Refinement, April 2026. URL http://arxiv.org/abs/2507.08390. arXiv:2507.08390 [cs] version: 3.

Tom Dillion. Tdoku: A fast sudoku solver and generator. https://t-dillon.github.io/tdoku/, 2025.

Zemin Huang, Yuhang Wang, Zhiyang Chen, and Guo-Jun Qi. Don't Settle Too Early: Self-Reflective Remasking for Diffusion Language Models, September 2025. URL http://arxiv.org/abs/2509.23653. arXiv:2509.23653 [cs].

Jaeyeon Kim, Seunggeun Kim, Taekyun Lee, David Z. Pan, Hyeji Kim, Sham Kakade, and Sitan Chen. Fine-Tuning Masked Diffusion for Provable Self-Correction, December 2025. URL http://arxiv.org/abs/2510.01384. arXiv:2510.01384 [cs].

Jaeyeon Kim, Jonathan Geuter, David Alvarez-Melis, Sham Kakade, and Sitan Chen. Stop Training for the Worst: Progressive Unmasking Accelerates Masked Diffusion Training, February 2026. URL http://arxiv.org/abs/2602.10314. arXiv:2602.10314 [cs].

Sanghyun Lee, Sunwoo Kim, Seungryong Kim, Jongho Park, and Dongmin Park. Effective Test-Time Scaling of Discrete Diffusion through Iterative Refinement, November 2025. URL http://arxiv.org/abs/2511.05562. arXiv:2511.05562 [cs] version: 1.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation. In *Advances in Neural Information Processing Systems*, 2023.

Liming Liu, Binxuan Huang, Zixuan Zhang, Xin Liu, Bing Yin, and Tuo Zhao. BackPlay: Plug-in Look-Back Self-Correction for Diffusion Language Models, February 2026. URL http://arxiv.org/abs/2601.06428. arXiv:2601.06428 [cs].

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. In *International Conference on Machine Learning*, pp. 32819–32848. PMLR, 2024.

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.

Zijing Ou, Chinmay Pani, and Yingzhen Li. Inference-Time Scaling of Discrete Diffusion Models via Importance Weighting and Optimal Proposal Design, March 2026. URL http://arxiv.org/abs/2505.22524. arXiv:2505.22524 [cs].

Prin Phunyaphibarn and Minhyuk Sung. Reward-Guided Discrete Diffusion via Clean-Sample Markov Chain for Molecule and Biological Sequence Design, February 2026. URL http://arxiv.org/abs/2602.09424. arXiv:2602.09424 [cs].

Subham S. Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T. Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and Effective Masked Diffusion Language Models. *Advances in Neural Information Processing Systems*, 37:130136–130184, December 2024. doi: 10.52202/079017-4135. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/eb0b13cc515724ab8015bc978fdde0ad-Abstract-Conference.html.

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

Yash Shah, Abhijit Chakraborty, Naresh Kumar Devulapally, Vishnu Lokhande, and Vivek Gupta. OSCAR: Orchestrated Self-verification and Cross-path Refinement, April 2026. URL http://arxiv.org/abs/2604.01624. arXiv:2604.01624 [cs].

Jiaxin Shi, Kehang Han, Zhe Wang, Arnaud Doucet, and Michalis Titsias. Simplified and generalized masked diffusion for discrete data. *Advances in neural information processing systems*, 37: 103131–103167, 2024.

Raghav Singhal, Zachary Horvitz, Ryan Teehan, Mengye Ren, Zhou Yu, Kathleen McKeown, and Rajesh Ranganath. A General Framework for Inference-time Scaling and Steering of Diffusion Models, July 2025. URL http://arxiv.org/abs/2501.06848. arXiv:2501.06848 [cs].

Atula Tejaswi, Litu Rout, Constantine Caramanis, Sanjay Shakkottai, and Sujay Sanghavi. EntRGi: Entropy Aware Reward Guidance for Diffusion Language Models, February 2026. URL http://arxiv.org/abs/2602.05000. arXiv:2602.05000 [cs].

Masatoshi Uehara, Xingyu Su, Yulai Zhao, Xiner Li, Aviv Regev, Shuiwang Ji, Sergey Levine, and Tommaso Biancalani. Reward-Guided Iterative Refinement in Diffusion Models at Test-Time with Applications to Protein and DNA Design, February 2025. URL http://arxiv.org/abs/2502.14944. arXiv:2502.14944 [cs].

Guan Wang, Jin Li, Yuhao Sun, Xing Chen, Changling Liu, Yue Wu, Meng Lu, Sen Song, and Yasin Abbasi Yadkori. Hierarchical reasoning model, 2025. URL https://arxiv.org/abs/2506.21734.

Guanghan Wang, Yair Schiff, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Remasking Discrete Diffusion Models with Inference-Time Scaling, February 2026. URL http://arxiv.org/abs/2503.00307. arXiv:2503.00307 [cs].

Shuibai Zhang, Fred Zhangzhi Peng, Yiheng Zhang, Jin Pan, and Grigorios G. Chrysos. Corrective Diffusion Language Models, January 2026. URL http://arxiv.org/abs/2512.15596. arXiv:2512.15596 [cs].

A Proofs

A.1 Proof for Proposition 1

Proof. By the definition of K_t,

$$\begin{aligned}(p_{\text{data}}K_t)(\mathbf{x}_0') &= \sum_{\mathbf{x}_0} p_{\text{data}}(\mathbf{x}_0) \sum_{\mathbf{x}_t} q_{0|t}(\mathbf{x}_0' \mid \mathbf{x}_t)\, q_{t|0}(\mathbf{x}_t \mid \mathbf{x}_0) \\ &= \sum_{\mathbf{x}_t} q_{0|t}(\mathbf{x}_0' \mid \mathbf{x}_t) \sum_{\mathbf{x}_0} q_{t|0}(\mathbf{x}_t \mid \mathbf{x}_0)\, p_{\text{data}}(\mathbf{x}_0) \\ &= \sum_{\mathbf{x}_t} q_{0|t}(\mathbf{x}_0' \mid \mathbf{x}_t)\, q_t(\mathbf{x}_t) \;=\; p_{\text{data}}(\mathbf{x}_0'),\end{aligned}$$

where the last step applies Bayes' rule $q_{0|t}(\mathbf{x}_0' \mid \mathbf{x}_t)\, q_t(\mathbf{x}_t) = q_{t|0}(\mathbf{x}_t \mid \mathbf{x}_0')\, p_{\text{data}}(\mathbf{x}_0')$ and then sums $q_{t|0}(\mathbf{x}_t \mid \mathbf{x}_0')$ over $\mathbf{x}_t$ to 1. □

A.2 Proof for Proposition 2

Proof. We first derive the stronger equality

$$\mathrm{KL}(p_{\text{data}} \,\|\, p') \;=\; \rho_t(p)\; \mathrm{KL}(p_{\text{data}} \,\|\, p) \;+\; \epsilon_t \;-\; R_t, \tag{3}$$

where $R_t \geq 0$ is the remainder defined below; The bound in Proposition 2 then follows by dropping R_t. Define joint distributions over $(\mathbf{x}_t, \mathbf{x}'_0)$

$$\mu(\mathbf{x}_t, \mathbf{x}'_0) \;=\; \tilde{q}_t(\mathbf{x}_t)\, f_\theta(\mathbf{x}'_0 \mid \mathbf{x}_t), \qquad\qquad \nu(\mathbf{x}_t, \mathbf{x}'_0) \;=\; q_t(\mathbf{x}_t)\, q_{0|t}(\mathbf{x}'_0 \mid \mathbf{x}_t).$$

The $\mathbf{x}'_0$-marginal of μ is p' (by definition of $p' = p\tilde{K}_t^\theta$), and the $\mathbf{x}'_0$-marginal of ν is p_{data} (by Bayes' rule, $q_t(\mathbf{x}_t)\, q_{0|t}(\mathbf{x}'_0 \mid \mathbf{x}_t) = p_{\mathrm{data}}(\mathbf{x}'_0)\, q_{t|0}(\mathbf{x}_t \mid \mathbf{x}'_0)$).

Expanding $\mathrm{KL}(\nu \,\|\, \mu)$ via the factorization $\mu = \tilde{q}_t \cdot f_\theta$ and $\nu = q_t \cdot q_{0|t}$ gives

$$\mathrm{KL}(\nu \,\|\, \mu) \;=\; \mathrm{KL}(q_t \,\|\, \tilde{q}_t) \;+\; \epsilon_t \;=\; \rho_t(p)\; \mathrm{KL}(p_{\mathrm{data}} \,\|\, p) \;+\; \epsilon_t,$$

using (1). Expanding the same divergence via the alternative factorization $\nu(\mathbf{x}_t, \mathbf{x}'_0) = p_{\mathrm{data}}(\mathbf{x}'_0)\, q_{t|0}(\mathbf{x}_t \mid \mathbf{x}'_0)$ yields

$$\mathrm{KL}(\nu \,\|\, \mu) \;=\; \mathrm{KL}(p_{\mathrm{data}} \,\|\, p') \;+\; R_t,$$

where

$$R_t \;=\; \mathbb{E}_{\mathbf{x}'_0 \sim p_{\mathrm{data}},\, \mathbf{x}_t \sim q_{t|0}(\cdot|\mathbf{x}'_0)} \left[\log \frac{q_{t|0}(\mathbf{x}_t \mid \mathbf{x}'_0)\, p'(\mathbf{x}'_0)}{\tilde{q}_t(\mathbf{x}_t)\, f_\theta(\mathbf{x}'_0 \mid \mathbf{x}_t)} \right] \;\geq\; 0,$$

the non-negativity following because R_t is itself a KL divergence. Equating the two expressions for $\mathrm{KL}(\nu \,\|\, \mu)$ gives (3); Dropping $R_t \geq 0$ yields the bound stated in Proposition 2. □

A.3 Proof for Proposition 3

Proof. Every masked sequence $\mathbf{x}_t \in (\mathcal{V} \cup \{\mathbf{m}\})^L$ is uniquely encoded as a pair $(S, \mathbf{x}^S)$, where S is the set of revealed positions and $\mathbf{x}^S \in \mathcal{V}^{|S|}$ are the revealed token values. Under both the independent and the uniform forward processes, the masking pattern S is drawn independently of $\mathbf{x}_0$, with marginal $\Pr(S)$ depending only on the noise schedule. Hence the joint distributions of $(S, \mathbf{x}^S)$ under q_t and $\tilde{q}_t$ are $q_t(S, \mathbf{x}^S) = \Pr(S)\, p^S_{\mathrm{data}}(\mathbf{x}^S)$ and $\tilde{q}_t(S, \mathbf{x}^S) = \Pr(S)\, p^S(\mathbf{x}^S)$, so $\Pr(S)$ cancels inside the log-ratio:

$$\mathrm{KL}(q_t \,\|\, \tilde{q}_t) \;=\; \sum_S \Pr(S) \sum_{\mathbf{x}^S} p^S_{\mathrm{data}}(\mathbf{x}^S) \log \frac{p^S_{\mathrm{data}}(\mathbf{x}^S)}{p^S(\mathbf{x}^S)} \;=\; \mathbb{E}_S \left[\mathrm{KL}(p^S_{\mathrm{data}} \,\|\, p^S) \right].$$

Dividing by $\mathrm{KL}(p_{\mathrm{data}} \| p)$ gives the first equality. For the second, apply the chain rule of KL to the partition $(\mathbf{x}^S, \mathbf{x}^{\bar{S}})$,

$$\mathrm{KL}(p_{\mathrm{data}} \,\|\, p) \;=\; \mathrm{KL}(p^S_{\mathrm{data}} \,\|\, p^S) \;+\; \mathbb{E}_{\mathbf{x}^S \sim p^S_{\mathrm{data}}} \left[\mathrm{KL}\Big(p_{\mathrm{data}}(\mathbf{x}^{\bar{S}} \mid \mathbf{x}^S) \,\Big\|\, p(\mathbf{x}^{\bar{S}} \mid \mathbf{x}^S) \Big) \right],$$

take expectation over S, and rearrange. □

A.4 Proof for Proposition 4

Proof. Starting from the marginal of $p' = p\tilde{K}_t^\theta$ on $\mathbf{x}_0$,

$$p'(\mathbf{x}_0) \;=\; \sum_{\mathbf{x}_t} \tilde{q}_t(\mathbf{x}_t)\, f_\theta(\mathbf{x}_0 \mid \mathbf{x}_t) \;=\; \mathbb{E}_{\mathbf{x}_t \sim \tilde{q}_t} \big[f_\theta(\mathbf{x}_0 \mid \mathbf{x}_t) \big],$$

apply Jensen's inequality directly:

$$-\log p'(\mathbf{x}_0) \;=\; -\log \mathbb{E}_{\mathbf{x}_t \sim \tilde{q}_t} \big[f_\theta(\mathbf{x}_0 \mid \mathbf{x}_t) \big] \;\leq\; \mathbb{E}_{\mathbf{x}_t \sim \tilde{q}_t} \big[-\log f_\theta(\mathbf{x}_0 \mid \mathbf{x}_t) \big].$$

Taking expectation over $\mathbf{x}_0 \sim p_{\mathrm{data}}$ and subtracting $H(p_{\mathrm{data}})$ yields the claim. □

A.5 Decoding with a General Sampling Operator

Proposition 2 and Proposition 4 are stated for a denoiser that is sampled from directly, whereas Algorithm 1 decodes with $\arg\max$ (Line 6) and Algorithm 2 reveals with $\arg\max$ (Line 9). The bounds are not tied to that choice.

Let $h \colon \Delta(\mathcal{V}) \to \Delta(\mathcal{V})$ be any *sampling operator*, applied independently at each position, and write $(h \circ f_\theta)(\cdot \mid \mathbf{x}_t) \in \Delta(\mathcal{V})$ for the resulting decoding distribution. Temperature sampling at temperature

μ is the operator that raises each probability to the power $1/\mu$ and renormalizes; Ordinary sampling is $\mu = 1$, and $\arg\max$ is the zero-temperature limit $\mu \to 0$. Replacing the denoiser by its decoded counterpart gives the *sampling-aware kernel*

$$\tilde{K}^{\theta}_{h,t}(\mathbf{x}'_0 \mid \mathbf{x}_0) \;=\; \sum_{\mathbf{x}_t} (h \circ f_\theta)(\mathbf{x}'_0 \mid \mathbf{x}_t)\, q_{t|0}(\mathbf{x}_t \mid \mathbf{x}_0),$$

which is what $\mathrm{R}^2\mathrm{D}$ actually iterates. Since nothing in the proof of Proposition 2 uses any property of f_θ beyond its being a conditional distribution over $\mathcal{V}^L$ given $\mathbf{x}_t$, the proof applies verbatim to $h \circ f_\theta$: For $p' = p\tilde{K}^{\theta}_{h,t}$,

$$\mathrm{KL}(p_{\mathrm{data}} \,\|\, p') \;\le\; \rho_t(p) \cdot \mathrm{KL}(p_{\mathrm{data}} \,\|\, p) \;+\; \epsilon_{h,t},$$
$$\epsilon_{h,t} \;=\; \mathbb{E}_{\mathbf{x}_t \sim q_t}\big[\mathrm{KL}\big(q_{0|t}(\cdot \mid \mathbf{x}_t) \,\big\|\, (h \circ f_\theta)(\cdot \mid \mathbf{x}_t)\big)\big].$$

The contraction coefficient $\rho_t(p)$ is unchanged, as it is a property of the masking channel alone and does not involve the model.

What the choice of h changes is the denoising error $\epsilon_{h,t}$. On practical tasks, including Sudoku, 3-SAT, and code generation, the posterior $q_{0|t}$ is almost deterministic, since the completion is usually unique given the revealed context. Sharpening f_θ toward its mode therefore makes $\epsilon_{h,t}$ smaller, not larger, wherever the model's prediction is correct. In this regime, $\arg\max$ tightens the error term rather than weakening it, and it cannot get stuck at a wrong fixed point, because the remasking step remains random: A mistaken token is exposed to remasking at the next kernel step.

Training is unaffected: We fit the soft f_θ, not $h \circ f_\theta$, with the cross-entropy objectives of Proposition 4 and Equation (2). It is standard practice to train at temperature 1 and decode at a lower temperature at inference.

B Trajectory Correction Training Algorithm

Algorithm 2 gives the pseudo-code of a single TCT update step described in Section 4. The B chains in a batch are advanced in a streaming fashion: After each gradient step, a chain that has reached the stage budget K is refilled with a fresh sample and a fully masked sequence, while every other chain reveals the top-k most confident masked positions with the model's $\arg\max$ predictions.

Algorithm 2 Trajectory Correction Training (Single Update Step)

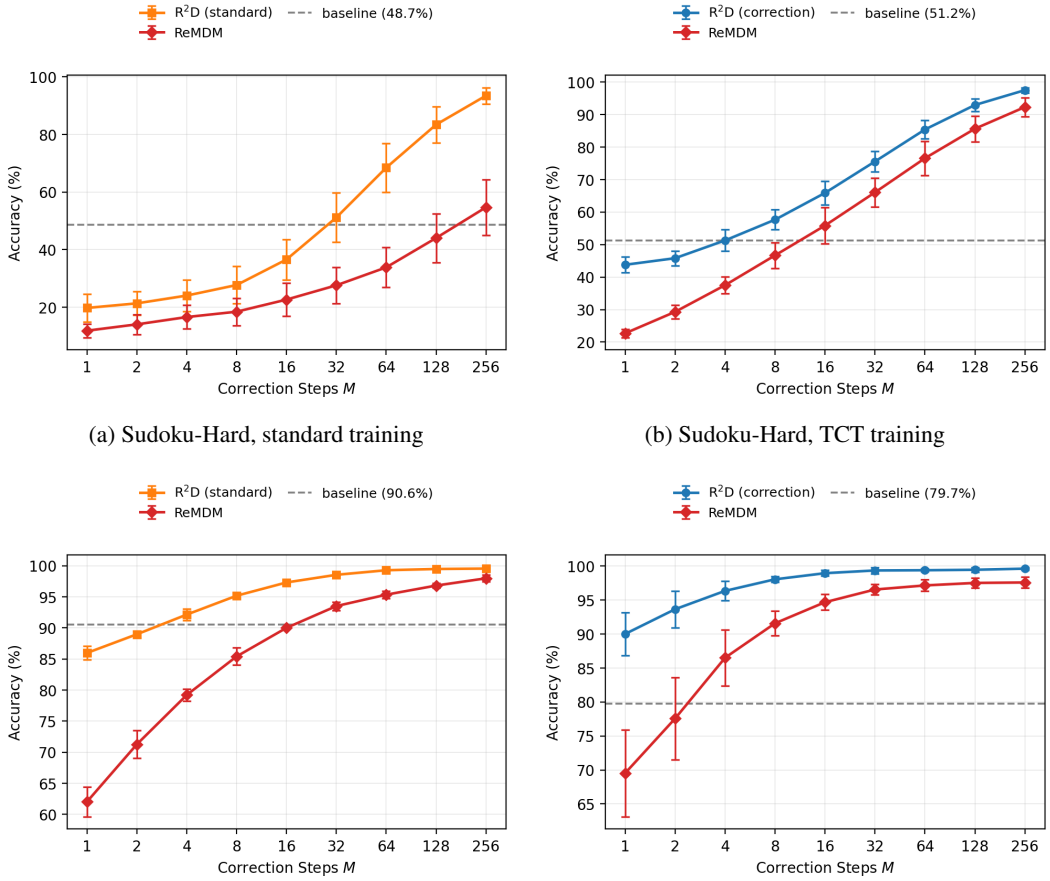
Require: MDM f_θ, max stages K, batch of partially masked sequences $\{\tilde{\mathbf{x}}^{(j)}\}_{j=1}^{B}$, paired clean samples $\{\mathbf{x}_0^{(j)}\}_{j=1}^{B}$, per-example stage counters $\{n^{(j)}\}_{j=1}^{B} \in [0, K]$

Compute logits $\{f_\theta(\cdot \mid \tilde{\mathbf{x}}^{(j)})\}_{j=1}^{B}$
Compute correction loss $L_{\mathrm{correct}}(\theta)$ (Equation (2)) from logits and $\{\mathbf{x}_0^{(j)}\}$; Update θ
for $j = 1, \cdots, B$ **in parallel do** ▷ Streaming update
  if $n^{(j)} = K$ **then**
    Sample $\mathbf{x}_0^{(j)} \sim p_{\mathrm{data}}$
    $n^{(j)} \leftarrow 0$
    Initialize $\tilde{\mathbf{x}}^{(j)} = \mathbf{m}^L$. ▷ refill: Restart chain on a new sequence
  else
    $\hat{x}_i^{(j)} \leftarrow \arg\max_{v \in V} f_\theta^i(v \mid \tilde{\mathbf{x}}^{(j)})$,
    $s_i \leftarrow \max_{v \in V} f_\theta^i(v \mid \tilde{\mathbf{x}}^{(j)})$ for masked positions i ▷ unmasking policy from the model
    Select reveal set $S^{(j)}$ from $\{s_i\}$ ▷ top-k with thresholding
    Unmask $\tilde{\mathbf{x}}_i^{(j)} \leftarrow \hat{x}_i^{(j)}$ for all $i \in S^{(j)}$; $n^{(j)} \leftarrow n^{(j)} + 1$ ▷ fill with model predictions
  end if
end for

C Additional Experiment Results

C.1 Error bars of R^2D and ReMDM inference.

Figure 4 shows the error bars of R^2D and ReMDM, under both TCT and standard training. We use 3 random seeds for training, and 3 random seeds for inference on each training checkpoint. The full figure below places those panels beside their TCT-training counterparts and draws the spread across training seeds.



(a) Sudoku-Hard, standard training

(b) Sudoku-Hard, TCT training

(c) 3-SAT, standard training

(d) 3-SAT, TCT training

Figure 4: Inference scaling on Sudoku-Hard (top) and 3-SAT (bottom), under standard training (left) and TCT training (right). The y-axis reports task accuracy and the x-axis the per-token correction budget M on a logarithmic scale; error bars are one standard deviation across three training seeds. The horizontal dashed line marks each panel's own no-correction baseline. R^2D scales near-log-linearly in all four panels and stays above ReMDM at every budget. Protocol details are in Section E.

C.2 Compute and Latency

We report the averaged NFE and averaged wall time of ReMDM, R^2D and ProSeCo. Table 2 is the compute-normalized view of Table 1, discussed in Section 5.4. We also report standard deviation of the accuracy for ReMDM and R^2D, under 3 random seeds. Wall time is measured at batch size 1 on a single NVIDIA GH200. The cost of correction relative to standard decoding is nonetheless real. At the largest budget, R^2D's per-sample latency is about $77\times$ that of standard decoding on the same checkpoint for Sudoku-Hard (49.4 s against 0.64 s) and about $26\times$ on 3-SAT (4.76 s against 0.18 s). What makes this practical is that the budget M is a free knob rather than a fixed cost: At $M = 1$ the

same sampler costs 1.31 s and 0.36 s respectively, within roughly twice the standard baseline, and every point in between is available along the curves of Figure 1.

Table 2: Compute-normalized comparison on Sudoku-Hard and 3-SAT. Each inference method is reported with its accuracy (in %), averaged NFE, and per-sample wall time at batch size 1 on a single NVIDIA GH200 GPU. R^2D is the only method that exits early: on Sudoku-Hard it reaches **97.5** in 49.4 s/sample where ReMDM reaches 92.3 in 187.9 s/sample — 5.2 points better at 3.8× less wall time. Measurement protocol in Section E.

		Sudoku-Hard			3-SAT		
Training method	Inference method	Acc.	NFE	s/sample	Acc.	NFE	s/sample
Standard	Standard	$48.7_{\pm 1.0}$	81	0.74	$90.6_{\pm 2.3}$	20	0.16
	ReMDM (standard)	$54.7_{\pm 9.7}$	20737	177.2	$93.5_{\pm 0.7}$	641	5.30
	R^2D (standard, $M = 1$)	$19.7_{\pm 4.7}$	162	1.41	$86.0_{\pm 1.1}$	40	0.42
	R^2D (standard, $M = M^\dagger$)	$93.5_{\pm 2.8}$	4607	36.9	$98.5_{\pm 0.2}$	336	2.70
ProSeCo	ProSeCo ($M = M^\dagger$)	55.7	20817	168.5	82.2	660	5.02
TCT (ours)	Standard	$51.2_{\pm 1.5}$	81	0.64	$79.7_{\pm 2.1}$	20	0.18
	ReMDM (correction)	$92.3_{\pm 2.9}$	20737	187.9	$96.5_{\pm 0.8}$	641	4.95
	R^2D (correction, $M = 1$)	$43.7_{\pm 2.4}$	162	1.31	$90.0_{\pm 3.2}$	40	0.36
	R^2D (correction, $M = M^\dagger$)	$\mathbf{97.5}_{\pm 0.8}$	5742	49.4	$\mathbf{99.3}_{\pm 0.5}$	343	4.76

C.3 Compute and Latency on HumanEval+

Table 3 pairs the accuracy of Figure 2 with the cost of the three correction methods. R^2D reaches top accuracy with roughly half of ReMDM's compute, resulting from the early-exit mechanism.

Table 3: HumanEval+ accuracy, averaged NFE and averaged wall time (in *minute* per sample) for the three iterative correction methods of Figure 2, all evaluated with the off-the-shelf LLaDA2.1-mini. R^2D reaches the top accuracy while spending roughly half of ReMDM's compute.

Method	Accuracy	Avg NFE	Avg wall time
R^2D (correction)	**85.4**	757	6.6
ProSeCo	81.7	692	6.0
ReMDM	71.6	1632	14.3

C.4 ReMDM Implementation Choice under TCT Training

We stick to the official implementation from `https://github.com/kuleshov-group/remdm` when running ReMDM with standard training. When the model is trained with TCT, we make one adaptation to ReMDM inference: We use argmax sampling instead of the categorical stochastic sampling in the official implementation.

We actually consider two critical implementation choices for ReMDM: Carry-over unmasking vs. no carry-over, and argmax sampling against stochastic sampling. Table 4 compares the 4 configurations on both Sudoku-Hard and 3-SAT. Each entry is evaluated on 9 repeated runs: three TCT training checkpoints with different seeds, and three inference seeds for each checkpoint. Stochastic sampling is significantly weaker on both tasks. This is because TCT models are trained on argmax sampling trajectories, and thus stochastically revealed tokens are out-of-distribution for these models. Under argmax sampling, Sudoku-Hard is indifferent to the unmasking choice, but 3-SAT favors carry-over unmasking. Therefore, we report the accuracy of ReMDM under carry-over unmasking + argmax sampling in Table 1 to ensure a fair comparison with R^2D, and refer to this configuration as **ReMDM (correction)**. The official configuration, i.e., carry-over unmasking + stochastic sampling, is referred to as ReMDM (standard) and is still used for standard-trained models.

We further investigate why the two unmasking choices (carry-over vs. no carry-over) have different impacts on the two tasks. We find that the TCT-trained model on Sudoku holds a correct grid exactly

Table 4: ReMDM implementation variants on the TCT checkpoint, comparing carry-over denoising against the sampling operator, at $M = 256$ on Sudoku-Hard and $M = 32$ on 3-SAT. Accuracy is in %, mean $\pm$ standard deviation over three training seeds $\times$ three inference seeds. The sampling operator dominates: $\arg\max$ is the operator TCT trains under. Carry-over is close to free on Sudoku-Hard but decisive on 3-SAT. The standard checkpoint is not ablated; there we run the official carry-over + stochastic configuration.

Carry-over	Sampling	Sudoku-Hard	3-SAT
Yes	Stochastic	$40.0_{\pm 6.7}$	$9.9_{\pm 3.0}$
No	Stochastic	$46.9_{\pm 12.3}$	$1.4_{\pm 1.8}$
Yes	Argmax	$92.3_{\pm 2.9}$	$\mathbf{96.5}_{\pm 0.8}$
No	Argmax	$\mathbf{92.4}_{\pm 2.6}$	$67.7_{\pm 30.9}$

in place, whereas on 3-SAT it keeps rewriting a small fraction of an already-correct assignment. To test this, we feed the TCT-trained model the ground-truth answer, randomly mask a fraction of the answer tokens, run a single forward pass *without* carry-over, and measure the fraction of revealed tokens the model reproduces unchanged. As shown in Table 5, on Sudoku-Hard the model preserves every correct revealed cell at every masking ratio. On 3-SAT, however, preservation is exact at high mask ratios but decreases at smaller ones. In other words, the model tends to modify the correct tokens at a small probability when most of the tokens are revealed in the sequence. Therefore, a correct solution can be modified by mistake near the end of generation, and this explains the high variance on 3-SAT without carry-over.

Table 5: Token preservation of the TCT-trained denoiser: The fraction of correct revealed tokens that survive a single forward pass without carry-over, averaged over three training seeds. On Sudoku-Hard the correct grid is an exact fixed point of the denoiser at every masking ratio; On 3-SAT it is not, and preservation is weakest precisely when the sequence is nearly complete, which is why carry-over is essential there and not on Sudoku-Hard.

Mask ratio	Sudoku-Hard	3-SAT
0.00	1.000	0.951
0.10	1.000	0.988
0.30	1.000	0.997
0.50	1.000	0.998
0.70	1.000	1.000
0.90	1.000	1.000
0.95	1.000	1.000

D Ablation Study

D.1 R^2D Inference Design

This section ablates the three elements in R^2D design. Starting from the default configuration, each row changes one design element of Algorithm 1 in isolation: Uniform remasking is replaced by low-confidence remasking, early exit is disabled, and the $\arg\max$ decoding of Line 6 is replaced by sampling from the denoiser. Table 6 gives the accuracy result along with averaged NFE on Sudoku-Hard with TCT training, under different compute budgets. We use three model checkpoints trained with three different random seeds. Each table entry is measured on 9 repeated evaluations, three runs for each checkpoint.

Low-confidence remasking almost removes the scaling effect: Accuracy moves only from 68.7% to 69.6% across a $16\times$ increase in budget, where uniform remasking goes from 65.8% to 97.5% over the same sweep. Removing early exit preserves the scaling trend but pays heavily for it: At $M = 256$ the NFE is nearly 4 times that of the default configuration. Moreover, its accuracy is still lower than that of the default configuration, because random remasking can move the sequence away from the stationary point after convergence. Stochastic sampling loses 10.4 points on accuracy because TCT

trains the denoiser along an $\arg\max$ trajectory. It also spends more NFE, as a stochastic prediction makes it harder to fulfill the early-exit test.

Table 6: $\mathrm{R}^2\mathrm{D}$ (correction) inference design on Sudoku-Hard under TCT training. Each row changes one design element of Algorithm 1 relative to `Default`. Each cell reports accuracy (in $\%$, mean $\pm$ standard deviation over three training seeds) with averaged NFE in parentheses. Low-confidence remasking almost removes the scaling effect; Removing early exit preserves scaling but at markedly higher NFE; Stochastic sampling in place of $\arg\max$ costs accuracy at every budget while spending more NFE.

Run	$M = 16$	$M = 32$	$M = 64$	$M = 128$	$M = 256$
`Default`	65.8±3.6 (1050)	**75.6**±3.1 (1477)	**85.4**±2.8 (2412)	**92.9**±1.9 (3552)	**97.5**±0.8 (5742)
`Low confidence`	**68.7**±1.8 (751)	69.0±2.4 (990)	69.5±2.4 (1356)	69.3±2.4 (2212)	69.6±2.3 (3271)
`No early exit`	63.9±3.1 (1377)	71.7±3.3 (2673)	79.2±4.1 (5265)	86.7±3.9 (10449)	93.3±2.4 (20817)
`Stochastic`	37.5±4.3 (1181)	50.1±6.3 (2034)	63.6±6.0 (3005)	76.1±6.3 (4146)	87.1±4.7 (6156)

D.2 TCT Training Design

The $\arg\max$ reveal of Line 9 and the top-k confidence reveal of Line 11 in Algorithm 2 are deliberate design choices rather than arbitrary deviations from the theory. Following PUMA (Kim et al., 2026), TCT trains the denoiser on its own inference-time trajectory; since $\mathrm{R}^2\mathrm{D}$ decodes with $\arg\max$, revealing with $\arg\max$ during training keeps the training and inference operators consistent, and revealing the highest-confidence positions keeps the training trajectory close to correct rather than letting it degrade into noise.

Table 7 ablates each choice independently on Sudoku-Hard, replacing the $\arg\max$ reveal with temperature-1 sampling and, separately, the top-k reveal with a random choice among the masked positions. Position selection is the load-bearing choice: Random reveal costs 12.0 points. The reason is that top-k keeps the revealed prefix mostly correct, so the model spends its capacity learning to predict the remaining masks from a valid partial solution — that is, learning the task structure — rather than learning to complete corrupted grids. The reveal operator is a minor lever by comparison: Stochastic reveal costs only 0.7 points, consistent with its role as the operator that $\mathrm{R}^2\mathrm{D}$ inference matches. Both approximations therefore help rather than hurt.

Table 7: Reveal rule in TCT training (Algorithm 2) on Sudoku-Hard, replacing the argmax reveal of Line 9 and the top-k confidence reveal of Line 11. Position selection is the load-bearing choice, while argmax reveal is a minor lever; Both approximations help rather than hurt.

Reveal rule	Sudoku-Hard acc. ($\%$)
Top-k confidence + argmax (published)	**97.5**
Random position + argmax (Line 11 ablation)	85.5
Top-k confidence + stochastic (Line 9 ablation)	96.8

E Additional Experiment Details

E.1 Sudoku-Hard

Data link. `https://huggingface.co/datasets/sapientinc/sudoku-extreme`.

Model architecture. The denoiser is a small Qwen2-style decoder-only transformer trained from scratch. The full architecture is summarized in Table 8 (column "Sudoku-Hard"); The resulting parameter count is approximately 6.8M, of which the embedding and language-model head contribute only ~ 5.6K parameters due to the tiny vocabulary (digits 1–9, the mask token, and a padding token).

Table 8: Denoiser architecture for Sudoku-Hard and 3-SAT. The two share a Qwen2-style backbone and differ only in vocabulary size and maximum position, which are dictated by the input encoding.

Hyperparameter	Sudoku-Hard	3-SAT
Vocabulary size	11	42
Hidden size	256	256
FFN intermediate size (SwiGLU)	768	768
Number of layers	8	8
Number of attention heads	8	8
Number of key/value heads	8	8
Maximum position	162	657
Dropout	0.0	0.0
Total parameters	≈ 6.8M	≈ 6.8M

Training. The denoiser is trained with TCT (Algorithm 2) on the Sudoku-Hard training split. We use a per-GPU batch size of 256. The checkpoint reported in the main paper is taken at step $460{,}000$. A single training run takes around 3 hours on a single Nvidia L40 GPU.

Inference. We set correction frequency to 1 and early-exit patience to $s = 5$ consecutive unchanged $\hat{\mathbf{x}}_0$, run the inner loop at every grid point (no skipped large-t steps), and disable early commitment of revealed tokens so that the kernel may freely revise any position throughout the trajectory.

E.2 3-SAT

Data link. `https://www.cs.ubc.ca/~hoos/SATLIB/benchm.html`

Model architecture. The 3-SAT denoiser shares the same Qwen2-style backbone as the Sudoku-Hard model and differs only in vocabulary size and maximum context length, both of which are dictated by the SATLIB encoding (20 variables, 91 clauses). See Table 8 (column "3-SAT"); The parameter count is again approximately 6.8M.

Training. The denoiser is trained from scratch with TCT (Algorithm 2) on a 1M-instance synthetic 3-SAT training set, using $K = 5$ correction phases per outer trajectory. The checkpoint reported in the main paper is taken at step $100{,}000$. A single training run takes around 4 hours on a single Nvidia GH200 GPU.

Inference. As on Sudoku-Hard, we set correction frequency to 1 and early-exit patience to $s = 5$ consecutive unchanged $\hat{\mathbf{x}}_0$, run the inner loop at every grid point (no skipped large-t steps), and disable early commitment of revealed tokens so that the kernel may freely revise any position throughout the trajectory.

E.3 HumanEval+

Benchmark link can be found at `https://github.com/evalplus/evalplus`.

Unlike Sudoku-Hard and 3-SAT, this experiment is inference-only: We evaluate the off-the-shelf LLaDA2.1-mini (16B MoE) (Bie et al., 2026) as released, with no TCT and no finetuning of any kind, so what varies across methods is the inference algorithm alone. All 164 HumanEval+ problems (Liu et al., 2023) are scored with EvalPlus; generations are produced on the base HumanEval prompts and re-scored against the plus test suite.

LLaDA2.1-mini decodes a response in blocks rather than over the whole sequence at once: The response is split into consecutive blocks of 32 tokens, up to 512 generated tokens (16 blocks), and the model attends to the prompt and all previously decoded blocks while denoising only the current one. Within a block we decode greedily (temperature 0) with confidence-thresholded parallel unmasking: Every masked position whose top-token probability exceeds 0.9 is revealed at that step, with at least one position revealed per step so that decoding always advances. Given the block-diffusion nature of

the model, $\mathrm{R}^2\mathrm{D}$, ProSeCo and ReMDM all adopt a block-wise correction: Correction only happens on the current block, and prior blocks and future masks are left untouched. $\mathrm{R}^2\mathrm{D}$ uses correction frequency 1 (the kernel is applied at every block) and the early-exit patience is $\tau = 2$. Remasking is uniform at random, as in Algorithm 1. $\mathrm{R}^2\mathrm{D}$ is reported as the mean $\pm$ standard deviation over three inference seeds. $\mathrm{R}^2\mathrm{D}$ and ReMDM both use 32 correction steps per token. ProSeCo uses 32 correction steps per block.