LET IT GO OR LEARN TO SELF-CORRECT: CONTINUOUS DIFFUSION FOR CONSTRAINED DISCRETE TASKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Denoising Diffusion Probabilistic Models (DDPMs) generate samples by starting from noise and repeatedly denoising while keeping each update close to the current noisy state. This behavior is effective in many continuous domains, but its role is less clear for globally constrained discrete tasks, such as Sudoku, graph connectivity, Latin squares, and N-queens. In such settings, early discrete errors can be difficult to undo. As a result, standard diffusion sampling may preserve early mistakes, even when the model's clean predictions are informative. We compare standard samplers to sampling directly from the model's clean prediction. Without retraining, this single change improves Sudoku validity from 31% to 95%, with consistent gains across the other discrete tasks. We hypothesize that staying close to the current noisy state is harmful because the reverse trajectory can drift off the forward noising distribution the model was trained on. To reduce this train-test mismatch, we further introduce self-correction training, which exposes the model to its own predictions, improving robustness to errors that arise during inference. This substantially improves the performance of standard samplers. Our results suggest that continuous diffusion models can learn nontrivial global constraints, but discrete reasoning tasks require better alignment between training and inference: either through samplers that reduce commitment to early decisions, or through training that teaches the model to correct its own inference-time errors.

1 INTRODUCTION

Diffusion models have become a dominant paradigm for generative modeling in continuous domains, achieving state-of-the-art performance on high-dimensional perceptual data such as images (Ho et al., 2020; Karras et al., 2022; Song et al., 2020a;b; Rombach et al., 2022) and videos (Alonso et al., 2024; Ho et al., 2022; Xing et al., 2024). Given this success, it is worth asking whether they can also handle globally constrained discrete tasks. Prior work has incorporated constraints into diffusion models through explicit mechanisms: projection (Christopher et al., 2024; Utkarsh et al., 2026), guidance (Inoue et al., 2023), or search-based post-processing (Sun & Yang, 2023). Discrete diffusion methods built on D3PM (Austin et al., 2021) have been applied to combinatorial optimization (Sun & Yang, 2023), puzzle solving and satisfiability (Ye et al., 2025), and controlled text and molecule generation (Cardei et al., 2026). On combinatorial problems, DIFUSCO (Sun & Yang, 2023) evaluated both D3PM-based discrete diffusion and continuous diffusion (Chen et al., 2022), reporting the latter as less effective. In this work, we investigate why standard continuous diffusion sampling can fail on these tasks and whether changes to sampling and training can mitigate these failures.

We study constrained discrete tasks where constraints are implicit in the data rather than provided explicitly to the model: Sudoku (Jolicoeur-Martineau, 2025; Wang et al., 2025; Wewer et al., 2025), graph connectivity (Du et al., 2024), Latin squares, and N-queens. These benchmarks are closely related to recent iterative-reasoning approaches: HRM/TRM use recursive refinement (Wang et al., 2025; Jolicoeur-Martineau, 2025), SRM studies denoising-based spatial reasoning and sampling-order choices (Wewer et al., 2025), and IRED casts graph connectivity as iterative energy diffusion (Du et al., 2024). Our goal is complementary: we use these tasks to diagnose standard continuous diffusion itself, asking when the denoiser learns useful global structure, why inference can fail to produce valid samples, and how it can be improved.

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.

In DDPMs (Ho et al., 2020), each reverse step makes a small update conditioned on the current state, which is well suited to perceptual data but its effect on globally constrained discrete tasks is less clear. In constrained discrete tasks, we observe a failure mode of standard DDPM sampling: generated samples can look locally plausible (e.g. each cell resembles a valid symbol: one-hot encoding or valid MNIST image) but remain globally invalid (e.g. Sudoku constraints are not satisfied). This raises a question: does the denoiser fail to learn the constraints, or does the reverse update fail to use the denoiser's clean prediction effectively?

We first test this by modifying only the reverse update. Both DDPM and the modified sampler use the current state x_t as input to the denoiser, producing a clean proposal $\hat{x}_0(x_t, t)$. The difference is what happens after this proposal is formed: DDPM carries a direct x_t-dependent residual into the reverse mean, keeping the next state tied to the current analog state. We remove this direct residual, yielding a limiting case of the generalized DDIM/DDPM sampling family (Song et al., 2020a) that we call Tweedie reprojection. This inference-only change gives large gains on the same checkpoints, improving one-hot Sudoku validity from 31% to 95%. Thus, the denoiser has learned useful constraint structure, but not always robustly enough to guide the DDPM trajectory once that trajectory begins to drift.

We interpret this as a form of training-inference mismatch (Ning et al., 2023; 2024; Deng et al., 2022; Ren et al., 2024; Zhang et al., 2025; Li & van der Schaar, 2024). During training, the model sees forward-noised valid samples. During sampling, however, the reverse process is driven by the model's own imperfect predictions. Small denoising errors can move the trajectory toward states that are locally plausible but globally invalid; the DDPM residual can then keep subsequent updates close to these states, causing errors to compound. This motivates us to also modify the training process with self-correction loss, where the model is exposed to noised versions of its own intermediate predictions and trained to recover the original valid target. This training modification improves standard samplers: for example, DDPM's validity on one-hot Sudoku rises from 31% to 87%.

Our study focuses on controlled constrained-discrete benchmarks. Tweedie reprojection should be viewed as a diagnostic for decoded constraint satisfaction, not as a general sampler: in perceptually rich domains, the x_t residual may carry important detail and discarding it may harm generation quality. Self-correction only partially addresses the training-inference mismatch; more complete approaches remain future work.

In summary, our contributions are:

- We identify a failure mode of continuous diffusion on constrained discrete reasoning tasks, where inference can lead to locally plausible but globally inconsistent states.
- We study the role of the direct x_t-dependent residual in DDPM sampling, and show that removing this residual can substantially improve constraint satisfaction without retraining.
- We propose a self-correction training procedure exposing the denoiser to noisy versions of its own predictions. This improves validity under standard samplers on the evaluated tasks.

2 PRELIMINARIES

We briefly review the diffusion notation used throughout the paper. We use $t = 0$ for clean data and $t = T$ for noise. In the DDPM formulation, $t \in \{0, 1, \ldots, T\}$ is discrete and $t - 1$ denotes one step toward data. For ODE/SDE samplers, t is continuous in $[0, T]$. We use the same symbols x_t, $\alpha(t)$, and $\beta(t)$ in both cases.

Forward process. Let $x_0 \in \mathbb{R}^d$ be a clean sample. We use the standard variance-preserving diffusion path

$$x_t = \alpha(t)x_0 + \beta(t)\epsilon, \qquad \epsilon \sim \mathcal{N}(0, I), \qquad \beta(t) \doteq \sqrt{1 - \alpha(t)^2}, \tag{1}$$

where $x_t \in \mathbb{R}^d$ is the resulting noisy sample, $\alpha(0) = 1$ and $\alpha(T) \approx 0$. Thus $q(x_t \mid x_0) = \mathcal{N}(\alpha(t)x_0, \beta(t)^2 I)$. The equivalent discrete Markov chain realization is given in Appendix A.

DDPM reverse process. In the discrete formulation, the forward Markov chain gives a tractable Gaussian posterior

$$q(x_{t-1} \mid x_t, x_0) = \mathcal{N}\Big(\mu_t(x_t, x_0), \tilde{\beta}_t I\Big), \tag{2}$$

with closed-form mean and variance given in Appendix A. Since x_0 is unknown at sampling time, DDPM replaces it with the model's current clean-sample estimate:

$$p_\theta(x_{t-1} \mid x_t) \doteq q(x_{t-1} \mid x_t, \hat{x}_0(x_t, t)). \tag{3}$$

In other words, each DDPM reverse step conditions on both the current noisy state x_t and the model's current clean guess $\hat{x}_0(x_t, t)$.

Training objective and parameterization. Diffusion models can be written in several equivalent parameterizations, including clean-sample prediction, noise prediction, score prediction, and velocity prediction (Ho et al., 2020; Song & Ermon, 2019; Lipman et al., 2022; 2024). At the optimum, these quantities can be converted between one another; the conversion formulas are given in Appendix A. Following Li & He (2026), the prediction parameterization and the loss target are separate design choices. In this work, we use x-prediction and train directly against the clean sample:

$$\mathcal{L}_{\text{simple}}(\theta) = \mathbb{E}_{x_0,t,\epsilon}\left[\|f_\theta(x_t, t) - x_0\|^2\right], \qquad \hat{x}_0(x_t, t) \doteq f_\theta(x_t, t). \tag{4}$$

Under that loss, the optimal x-prediction is $\mathbb{E}[x_0 \mid x_t]$. By Tweedie's formula (Efron, 2011),

$$\mathbb{E}[x_0 \mid x_t] = \frac{x_t + \beta(t)^2 \nabla_{x_t} \log q_t(x_t)}{\alpha(t)}, \tag{5}$$

where q_t is the marginal distribution of x_t. We therefore interpret $\hat{x}_0(x_t, t)$ as a plug-in Tweedie estimate of the clean sample.

Continuous-time samplers and noise scale. For continuous-time samplers, the same path can be described by a marginal vector field $u_t(x)$, yielding the probability-flow ODE $\mathrm{d}x = u_t(x)\,\mathrm{d}t$.

More generally, for any non-negative diffusion schedule $\sigma(t)$, the reverse-time SDE

$$\mathrm{d}x = \left[u_t(x) - \frac{\sigma(t)^2}{2}\nabla_x \log q_t(x)\right]\mathrm{d}t + \sigma(t)\,\mathrm{d}w_t \tag{6}$$

shares the same marginals q_t (Holderrieth & Erives, 2025, Thm. 17). Thus $\sigma(t)$ changes the stochasticity of the sampler without changing the target probability path theoretically. The probability-flow ODE is recovered by setting $\sigma(t) = 0$, while the variance-preserving reverse SDE of Song et al. (2020b) corresponds to $\sigma(t)^2 = -2\dot{\alpha}(t)/\alpha(t)$. In practice, learned denoisers and finite-step solvers introduce model and discretization error, so different choices of $\sigma(t)$ can lead to different empirical performance. We therefore treat $\sigma(t)$ as a sampler hyperparameter for Euler-Maruyama-based samplers.

3 METHOD

3.1 INFERENCE UNDER CONSTRAINT VIOLATIONS

Under the denoising objective in Eq. 4, the model is only exposed to noisy versions x_t of valid samples x_0. The learned denoiser $f_\theta(x_t, t)$ is then used as a plug-in estimate of the clean object in reverse updates of the form $q(x_{t-1} \mid x_t, \hat{x}_0)$. However, at inference time on constrained tasks, during sampling, imperfect predictions can produce a state distribution that differs from the forward-noising distribution used for training.

This creates a mismatch between the states encountered in training and those visited during inference (exposure bias). In this regime, the standard DDPM reverse update $q(x_{t-1} \mid x_t, \hat{x}_0)$ can preserve information from the current state even when that state contains wrong discrete commitments, leading to global constraint violations. We address this mismatch from two directions: (i) a sampling rule that reduces direct dependence on the current state after forming the clean proposal $\hat{x}_0$ (Sec. 3.2), and (ii) a training objective that exposes the model to its own predictions, improving robustness to the states encountered during sampling (Sec. 3.3).

3.2 A STOCHASTIC ANCHOR-FREE UPDATE

First, we investigate the simplified reverse update that reduces the influence of the current state x_t.

We denote $\hat{x}_0 \equiv \hat{x}_0(x_t, t)$, and set $B_t = \sqrt{\delta(t)}\,(1-\alpha(t-1)^2)/(1-\alpha(t)^2)$ where $\delta(t) = \frac{\alpha(t)^2}{\alpha(t-1)^2}$. The DDPM ancestral update can then be written as (see Eq. 16)

$$p_\theta^{\mathrm{DDPM}}(x_{t-1} \mid x_t) = \mathcal{N}\Big(\underbrace{\alpha(t-1)\hat{x}_0}_{\text{Tweedie center}} + \underbrace{B_t\big(x_t - \alpha(t)\hat{x}_0\big)}_{x_t\text{-dependent residual}},\ \underbrace{(1-\alpha(t-1)^2)}_{\text{marginal variance}}\ \underbrace{\frac{1-\delta(t)}{1-\alpha(t)^2}}_{\text{shrinking factor}}\ I\Big). \quad (7)$$

We investigate the limiting case with no direct dependence on x_t in the reverse step. We refer to this update as TWEEDIE REPROJECTION:

$$p_\theta^{\mathrm{Tw}}(x_{t-1} \mid x_t) = \mathcal{N}\big(\alpha(t-1)\hat{x}_0, (1-\alpha(t-1)^2)I\big)\,. \quad (8)$$

This update removes the direct x_t-dependent residual from the reverse mean and re-noises the clean prediction using the forward marginal variance. Thus, x_t is only used through the denoiser prediction $\hat{x}_0(x_t, t)$. Tweedie reprojection is an endpoint of the generalized DDIM sampler from Song et al. (2020a) (see Appendix B.1). We use it as an inference-time diagnostic to test whether constraint satisfaction has already been learned by the denoiser, but is not preserved by the standard sampler.

3.3 SELF-CORRECTION TRAINING

To improve constraint satisfaction of the final sample, we introduce SELF-CORRECTION, a training procedure that exposes the model to its own imperfect predictions.

Illustrated in Figure 1, given a clean sample x_0, we first sample a noise level $t_1 \sim \mathcal{U}[0, T]$ and construct x_{t_1} as usual. We then compute an intermediate prediction $\hat{x}_0 = f_\theta(x_{t_1}, t_1)$.

This prediction can be imperfect and may violate constraints, especially when t_1 is far from data. We next noise $\hat{x}_0$ (with gradients disabled) at a second level $t_2 \sim \mathcal{U}[0, t_1]$ to obtain $\tilde{x}_{t_2} = \alpha(t_2)\hat{x}_0 + \beta(t_2)\epsilon_2$. The model is trained to recover the original x_0 from $\tilde{x}_{t_2}$ using the loss $\mathcal{L}_{\mathrm{rec}} = \mathbb{E}[\|f_\theta(\tilde{x}_{t_2}, t_2) - x_0\|^2]$. This procedure generalizes naturally to multiple prediction steps, though longer unrolls were unstable when trained from scratch. A warm-started multi-step variant trained successfully but did not outperform one-step self-correction; see Appendix I. The full training objective combines this recovery loss with the usual denoising objective $\mathcal{L}_{\mathrm{simple}}$ from equation 4:

$$\mathcal{L}_{\mathrm{SC}} = \mathcal{L}_{\mathrm{rec}} + \lambda_{\mathrm{simple}}\mathcal{L}_{\mathrm{simple}}, \quad \lambda_{\mathrm{simple}} \geq 0. \quad (9)$$

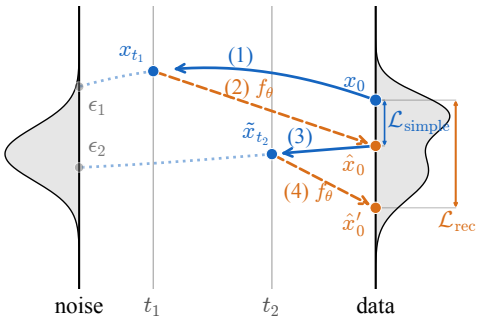


Figure 1: Self-correction training: Combining $\mathcal{L}_{\mathrm{rec}}$ and $\mathcal{L}_{\mathrm{simple}}$, training exposes the model to noisy versions of its own imperfect predictions, partially addressing the training-inference mismatch by encouraging it to correct constraint violations instead of only reinforcing locally consistent structure.

Here, $\mathcal{L}_{\mathrm{simple}}$ acts as an anchor to the original diffusion objective, preserving single-step denoising on forward-noised data while $\mathcal{L}_{\mathrm{rec}}$ trains correction from self-induced states. Algorithm 2 summarizes the training procedure.

For conditional generation, known values are pinned to their ground-truth after each step. Details on the pinning procedure for training and inference are provided in Appendices C and B.4, respectively.

4 EXPERIMENTS

We train and evaluate on the following benchmarks for constrained data generation and inpainting/completion with a focus on discrete-space reasoning tasks: Sudoku, Sudoku-Extreme, graph connectivity (GC), Latin squares, and N-queens (see Appendix D for details). For Sudoku, we use completed boards from Sudoku-Extreme and randomly generate conditioning masks, evaluating

on 21-clue puzzles and the Medium and Hard clue-count settings of SRM (Wewer et al., 2025). For Sudoku-Extreme (Jolicoeur-Martineau, 2025; Wang et al., 2025), we instead use the original masks guaranteeing a unique solution for each puzzle. Following IRED (Du et al., 2024), we train GC on $N = 12$ and evaluate on $N = 12$ and $N = 18$. We additionally evaluate Latin squares ($N = 7$) and N-queens ($N = 14$) under random in-painting and unconditional generation.

As we use continuous diffusion models, we lift discrete configurations into a continuous space (Chen et al., 2022) by encoding each token as a one-hot vector (see Appendix D.1). For all tasks except MNIST Sudoku (Wewer et al., 2025) and Graph Connectivity (GC) (Du et al., 2024), our network f_θ is a Transformer (Vaswani et al., 2017) with shared hyperparameters (see Appendix E.2 for architecture details). For Graph Connectivity, we adopt Neural Logic Machine-based (Dong et al., 2019) architecture of IRED (Du et al., 2024, Table 11) with their hyperparameters. As for MNIST Sudoku, we use the pre-trained SRM model (Wewer et al., 2025) without any additional training.

Results are compared across different samplers: DDPM, Euler, Euler-Maruyama (EM), EM decay (EM with decaying σ_t; equation 6), and our Tweedie reprojection sampler. For EM, we tune a constant noise level σ separately for each configuration. We first select a checkpoint step, based on the validity rate on the validation set under deterministic Euler sampling. At this step, we partition the validation set into five folds and select a single σ, by maximizing the pass@1 validity rate averaged over all folds. The fixed (step, σ) setting is then evaluated on the test set for each seed, and we report mean $\pm$ std across the three seeds. For EM decay, we jointly tune $(\sigma, t_{\text{decay}})$ using the same procedure. The sampler uses the selected noise level before t_{decay} and linearly anneals it to zero toward the data endpoint. See Appendix F for additional details on training and inference hyperparameters.

Results on discrete constrained tasks. Table 1 compares the continuous samplers within each training regime. The D3PM rows use separately trained discrete-diffusion models and are included as an external reference (Appendix E.1). We first consider the baseline models, trained without self-correction. The standard samplers, DDPM and EM, give broadly similar performance across tasks, with DDPM requiring no sampler hyperparameter tuning and EM using the noise level chosen according to the protocol described above. We then compare against two ways of changing the stochastic reverse process: EM decay, which allows larger noise early in sampling and anneals it toward the data endpoint, and Tweedie reprojection, which changes the reverse-step center by removing the x_t residual. Both modifications improve over the standard samplers under baseline, while Tweedie reprojection gives the largest and most consistent gains. For example, on 21-clue Sudoku, success improves from about 31% with DDPM to 95% with Tweedie reprojection. Similar improvements appear across other datasets. This suggests that, for one-hot encoded constrained discrete tasks, a model trained with the standard denoising objective can already make useful clean predictions.

The second block of Table 1 shows the effect of self-correction training. Self-correction substantially improves DDPM and EM, narrowing the gap between standard samplers and Tweedie reprojection. For instance, DDPM rises from about 31% to 87% on 21-clue Sudoku and from about 19% to 64% on Sudoku-Extreme pass@10. This suggests that self-correction partially fixes the reverse-process failure by training the model on sampler-induced states. Finally, combining self-correction with the modified samplers gives the strongest overall results: Tweedie reprojection and EM decay are both highly competitive, with each being best or tied for best on different tasks. Overall, the table supports two conclusions: removing the x_t residual is already highly beneficial at inference time for constrained discrete tasks, and self-correction further improves robustness by training the model to recover from errors encountered during sampling.

Table 1 focuses on validity, a strict metric under which a sample fails if any constraint is violated. To provide a more graded view of the generated solutions, we report two additional diagnostics in Appendix L. Table 11 measures how close the final continuous state is to its nearest one-hot configuration, while Table 12 reports the fraction of predicted cells involved in constraint violation. Together, these metrics help distinguish failures of discrete commitment from samples that are locally plausible but still violate a small number of global constraints.

D3PM's confidence-based and remasking samplers perform strongly on standard Sudoku, conditioned N-queens, and Latin squares. On Sudoku-Extreme, our best self-corrected continuous configurations achieve higher validity than the D3PM samplers evaluated here. IRED (Du et al., 2024) with the same architecture as ours reports 99.1% and 93.8% on graph connectivity at $N = 12$ and $N = 18$.

Table 1: **Success rates across samplers and tasks.** Self-correction training uses $\lambda_{\text{simple}} = 0.1$. Entries report mean ± standard deviation over three independently trained seeds, each evaluated with an independent test seed. Bold values indicate the best result for each configuration separately.

Sampler	Sudoku			Sudoku-Extreme		N-Queens		Latin		GC	
	21 clues	Medium	Hard	pass@1	pass@10	random	gen	random	gen	$N = 12$	$N = 18$
Baseline (no self-correction loss)											
DDPM	.31±.02	.83±.01	.38±.00	.05±.00	.19±.01	.53±.00	.06±.01	.71±.03	.87±.03	.80±.01	.67±.01
Euler	.20±.01	.78±.00	.23±.00	.04±.00	.13±.01	.45±.02	.04±.01	.58±.02	.75±.03	.74±.01	.58±.02
EM	.21±.01	.76±.01	.28±.00	.04±.00	.13±.01	.45±.02	.04±.00	.59±.02	.79±.07	.73±.00	.56±.02
EM decay	.81±.03	.96±.01	.80±.02	.18±.02	.58±.03	.69±.01	.15±.03	.96±.02	.99±.01	.94±.01	.89±.03
Tweedie reproj.	**.95±.01**	**.99±.00**	**.93±.01**	**.26±.01**	**.64±.02**	**.83±.00**	**.25±.01**	**.99±.01**	**1.00±.00**	**.97±.00**	**.95±.00**
Self-correction loss (with $\lambda_{\text{simple}} = 0.1$)											
DDPM	.87±.02	.96±.00	.83±.02	.22±.01	.64±.04	.74±.04	.26±.04	.98±.01	.92±.04	.88±.03	.77±.05
Euler	.79±.02	.94±.00	.67±.01	.22±.02	.61±.03	.68±.02	.14±.04	.94±.02	.64±.07	.84±.03	.68±.04
EM	.82±.02	.93±.01	.73±.03	.21±.02	.61±.03	.66±.02	.22±.04	.97±.02	.78±.07	.81±.08	.68±.04
EM decay	**.98±.00**	**.99±.00**	**.98±.00**	**.53±.04**	**.90±.02**	.81±.03	**.47±.04**	**1.00±.00**	**1.00±.00**	.88±.02	.80±.04
Tweedie reproj.	**.98±.01**	**.99±.00**	.94±.01	.49±.01	.84±.02	**.89±.01**	.35±.06	**1.00±.00**	.98±.02	**.99±.01**	**.99±.01**
Discrete diffusion (D3PM) reference											
Ancestral	.57±.05	.88±.02	.52±.04	.03±.01	.10±.03	.85±.01	.28±.01	.88±.03	.65±.05	.46±.00	.29±.02
Confidence	**1.00±.00**	**1.00±.00**	**.99±.01**	**.24±.02**	**.44±.05**	**.97±.00**	**.48±.01**	**1.00±.00**	**.99±.00**	**.99±.02**	**1.00±.00**
Remask	.99±.00	**1.00±.00**	.98±.00	.23±.02	.42±.03	**.97±.00**	**.48±.01**	**1.00±.00**	.96±.01	**.99±.02**	**1.00±.00**

Together, these results suggest that continuous diffusion remains promising for constrained discrete tasks once the training-inference mismatch is addressed.

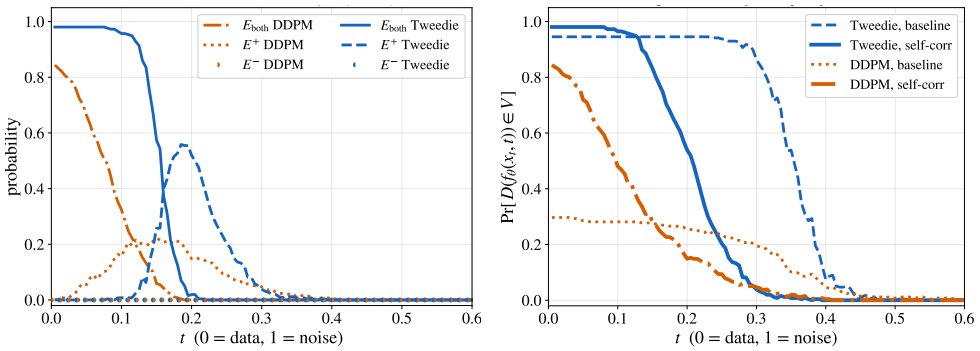
MNIST-Sudoku. As an inference-only sanity check beyond one-hot grids, we evaluate Tweedie reprojection on the MNIST-Sudoku Hard split of SRM (Wewer et al., 2025), where symbols are represented as MNIST images (LeCun, 1998) but success is still Sudoku validity. We use the released SRM diffusion-baseline checkpoint and change only the inference sampler without retraining. The original rectified flow baseline achieves $\mathbf{0.8}\%$ accuracy, while Tweedie reprojection raises accuracy to $\mathbf{65.7}\%$ (N=1000, 95% CI ±3 pp), exceeding the best SRM sampling-order strategy at 51.6%.

To additionally test whether our conclusions depend on one-hot representations or argmax decoding, we train our models using alternative continuous representations and decoders (Appendix K). We replace one-hot vectors with analog-bit codes and threshold decoding, fixed random embeddings and nearest-neighbour decoding, and mini MNIST-Sudoku with a learned CNN decoder. Across four representations in the single-seed experiments, baseline DDPM validity of $0.31, 0.19, 0.26, 0.01$ increases to $0.94, 0.89, 0.89, 0.42$ with Tweedie reprojection and to $0.87, 0.79, 0.76, 0.39$ with self-correction. In addition to the self-correction ablations in Appendix I, we report rectified-flow and consistency-model experiments in Appendices I.3 and I.4.

5 Analysis

Locally plausible but globally invalid states. The results above show that the same model can behave very differently under different samplers. The issue is not that the model never learns valid local symbols. The final continuous output can be close to a one-hot representation without its decoded grid being valid (Table 11). Similarly, in MNIST-Sudoku, individual cells can resemble recognizable digits while the full board violates Sudoku constraints. This motivates examining a possible training-inference mismatch. Standard denoising trains on forward-noised valid objects, $x_t = \alpha(t)x_0 + \beta(t)\epsilon, x_0 \in \mathcal{V}$. During inference, states are generated by repeated model-dependent reverse updates. We hypothesize that standard forward-noising training provides insufficient supervision for some errors arising along these trajectories.

DDPM can preserve globally invalid decoded states. Equation 7 shows that the DDPM reverse center contains an x_t-dependent residual in addition to the denoiser's clean prediction. This residual is not intrinsically harmful: it keeps the reverse update close to the current analog state, which is appropriate when the current state lies on a reliable trajectory and contains details that should persist.



(a) (b)

Figure 2: **DDPM-Tweedie gap on Sudoku. (a)** Decoded-center comparison at matched sampling times: E_{both} means both centers are valid, E_+ means only the Tweedie center is valid, and E_- means only the DDPM center is valid. **(b)** Validity of denoiser proposals $D(f_\theta(x_t, t))$ along Tweedie and DDPM trajectories.

The problem is that x_t can already encode mistakes that violate discrete constraints, and the denoiser has not necessarily been trained to correct such sampler-induced states. The residual can then keep the update near a locally plausible but globally invalid decoded configuration.

Figure 2a compares Tweedie and DDPM centers computed from the same current state x_t and denoiser prediction $\hat{x}_0(x_t, t)$. We perform this comparison separately along trajectories generated by each sampler. We write E_{both} for the event that both centers decode to valid boards at the same reverse time, E_+ for the event that only Tweedie center is valid, and E_- for the event that only DDPM center is valid. The residual preserves center validity on E_{both}, restores it on E_-, and destroys it on E_+. Across trajectories, E_+ is much larger than E_-. In this regime, DDPM residual more often turns a valid clean proposal into an invalid center than it rescues an invalid proposal. Appendix H.3 contains the corresponding analysis and additional trajectory diagnostics. Tweedie reprojection is limited by proposal quality: it cannot compensate when the denoiser's proposals are poor. Proposal quality is not the only source of failure: on Sudoku-Extreme, a valid proposal appeared at least once in 62% of failed Tweedie trajectories, but the final sample was still invalid (see Appendix M).

The sampler gap is not only a noise-scale effect. On the sampling side, EM decay shows that additional stochasticity can also help: it injects more noise early in sampling and anneals this noise near the data endpoint. This can break some bad intermediate commitments while still allowing the sample to settle near the end. However, Appendix Figure 6 shows that over the tested range, changing the sampling-noise multiplier does not close the baseline DDPM-Tweedie gap. Scaling the DDPM noise changes how broadly the kernel samples, but not where it is centered. Tweedie reprojection instead changes the center by sampling around the clean proposal rather than continuing the same state-dependent update. Increasing the number of DDPM steps also fails to close this gap in the evaluated Sudoku setting (Table 9).

Self-correction trains recovery from model-generated errors. The same mechanism suggests a training-side fix. Instead of exposing the model only to noisy valid objects as standard diffusion does, self-correction also exposes the model to noisy versions of its own intermediate predictions while keeping the original valid object as the target. Thus the model is trained to map model-induced, possibly invalid states back to a valid solution. Figure 2b shows this effect. Along DDPM trajectories, the baseline denoiser proposes valid grids much less often than along Tweedie trajectories. After self-correction training, proposal validity is higher along the resulting DDPM trajectories. This is consistent with improved recovery from model-generated errors, although the comparison changes both the denoiser and the states it visits.

Additional ablations support this interpretation (see Appendix I). The tested input perturbation (Ning et al., 2023) and self-conditioning (Chen et al., 2022) are weaker than self-correction, suggesting that the important ingredient is not generic robustness to extra noise or an additional memory channel, but exposure to model-induced states. In the random-symbol corruption ablation (see Appendix I.2),

DDPM validity increases from 34.5% to 64.0% with 20% corrupted cells and 56.6% with 40%, while self-correction reaches 78.2% (Table 7). Thus, recovery from model-generated errors outperforms the tested random-symbol corruptions. A complementary comparison finds closer agreement between self-correction inputs and DDPM states on the center-disagreement event E_+ (Appendix H.4).

6 Related work

Diffusion models and samplers. Score-based and diffusion models (Sohl-Dickstein et al., 2015; Ho et al., 2020; Song et al., 2020b;a; Karras et al., 2022; Lipman et al., 2024; Holderrieth & Erives, 2025; Lai et al., 2025) have achieved strong results in high-dimensional generation. Beyond standard DDPM sampling, alternative inference schemes include DDIM (Song et al., 2020a), standard ODE and SDE integrators (e.g. Euler, Heun, Euler-Maruyama) (Song et al., 2020b), stochastic predictor-corrector methods (Song et al., 2020b), and samplers with explicit Langevin-like noise injection steps to maintain correct marginals (Karras et al., 2022). Some methods also self-condition their models on noisy and estimated clean samples to improve sample quality (Chen et al., 2022; Watson et al., 2023). All these methods differ in their trade-off between stability and stochastic exploration, and in how much each update is anchored to the previous noisy sample versus relying on a fresh denoised prediction.

Inference-time control and constrained generation. A large body of work incorporates constraints or objectives during or after sampling. Methods include hard conditioning by masking or inpainting (Janner et al., 2022; Inoue et al., 2023; Mayet et al., 2025), Tweedie-based posterior sampling and guidance (Chung et al., 2023; 2022), projection onto constraint sets (Christopher et al., 2024; Cardei et al., 2026; Utkarsh et al., 2026), inference-time optimization (Utkarsh et al., 2026; Christopher et al., 2024; Cardei et al., 2026; Li et al., 2026b), guidance (Janner et al., 2022; Inoue et al., 2023), and search-based post-processing (Sun & Yang, 2023). These approaches typically assume access to explicit constraints or a constraint violation signal. In contrast, we consider the setting where constraints must be learned implicitly from data (Du et al., 2024), and study failure modes arising purely from inference dynamics.

Diffusion for structured and combinatorial tasks. Diffusion models have been applied to structured domains such as Sudoku, graphs, and combinatorial optimization (Sun & Yang, 2023; Du et al., 2024; Wewer et al., 2025; Avdeyev et al., 2023; Ye et al., 2025; Kim et al., 2025; Li et al., 2026a; Pereira, 2026). Discrete diffusion methods (Austin et al., 2021) and structured variants often outperform continuous diffusion (Chen et al., 2022) in such settings (Sun & Yang, 2023), highlighting the challenges of applying continuous diffusion to discrete-structured tasks. Continuous diffusion can be used by embedding discrete data in a continuous space (Hoogeboom et al., 2021; Chen et al., 2022; Avdeyev et al., 2023), and these representations can be restricted to a bounded support such as the probability simplex (Avdeyev et al., 2023). Our work explores standard continuous diffusion for highly structured generation and completion tasks.

Training-inference mismatch and exposure bias. Prior work has identified training-inference mismatch (exposure bias) as a source of error accumulation in sequential models (Bengio et al., 2015; Ross et al., 2011; Huang et al., 2026; Bachmann & Nagarajan, 2024; Chen et al., 2024; Du et al., 2024). In diffusion models, similar effects arise because the model is trained on noisy ground-truth samples but receives its own predictions as inputs at inference time (Ning et al., 2023; 2024; Deng et al., 2022; Ren et al., 2024; Zhang et al., 2025; Li & van der Schaar, 2024), and error has been shown to necessarily accumulate along the sampling trajectory of imperfect diffusion models under mild assumptions (Li & van der Schaar, 2024). Recent approaches mitigate exposure bias by modifying the training process: perturbing inputs (Ning et al., 2023), minimizing cumulative errors as regularization (Li & van der Schaar, 2024), or exposing the model to its own errors via truncated (Deng et al., 2022) or analytically simulated rollouts (Ren et al., 2024). Other methods mitigate diffusion exposure bias by modifying training objectives, adding correction modules, or rescaling predictions at inference time (Ren et al., 2024; Zhang et al., 2025; Ning et al., 2024). Whereas the diffusion-specific approaches cited above address exposure bias primarily in visual generation, we study its effect on decoded globally constrained discrete tasks.

Non-diffusion solvers for constrained tasks. OptNet (Amos & Kolter, 2017) and SATNet (Wang et al., 2019) present differentiable optimization-based solver layers. More recent work explores recursive refinement models for reasoning tasks (Wang et al., 2025; Jolicoeur-Martineau, 2025). Such models mitigate error compounding by training over rollouts of their refinement. Our work is complementary: we do not aim to compete with these solvers, but to characterize a fundamental limitation of continuous diffusion-based inference in such settings.

Recurrence and diffusion model. Concurrent work combines flow- or diffusion-based denoising with recurrence via self-conditioning (Helbling et al., 2026) or persistent hidden states (Drozdova et al., 2026; Suleymanzade et al., 2026). Tweedie reprojection can be seen as an implicit form of this recurrence: the clean estimate $\hat{x}_0$ is carried to the next step by re-noising alone, with no self-conditioning input or hidden-state channel and no retraining.

7 LIMITATIONS

Our experiments focus on constrained discrete tasks. Most tasks use one-hot encodings with argmax decoding, where validity is symbolic. MNIST-Sudoku, analog-bit and random-embedding experiments show that the similar validity gains appears beyond exact one-hot vectors, but the evaluation is still symbolic Sudoku validity. We therefore do not claim that Tweedie reprojection is appropriate for perceptually rich domains, where success also requires modeling a diverse continuous distribution over texture, geometry, color, and fine details.

Tweedie reprojection changes the nature of the reverse process. It uses x_t to form the proposal $\hat{x}_0 = f_\theta(x_t, t)$, but then discards the residual $x_t - \alpha(t)\hat{x}_0$, so it no longer enforces the same proximity between noisy states as DDPM posterior updates. This can help when the state contains wrong symbolic commitments, but in domains where meaningful variation exists within a decoded mode, or where fine continuous details are not fully captured by $\hat{x}_0$, repeated reprojection may lose information that a state-preserving sampler would retain.

Finally, self-correction only partially addresses the training-inference mismatch. It exposes the model to one-step model-induced states, improving training coverage, but longer sampling trajectories can still visit states not well represented during training. A more complete solution may require new training objectives or noise processes that better cover both forward-noised data and the sampler-induced states, which we leave for future work.

8 CONCLUSION

We studied continuous diffusion models for constrained discrete tasks represented in continuous space. Across Sudoku, graph connectivity, Latin squares, and N-queens, the same trained denoiser can behave very differently under different reverse processes. This reveals a training-inference mismatch: standard denoising trains on forward-noised valid objects, while inference sampling can create locally plausible but globally invalid states that are not well covered by the training distribution. Tweedie reprojection exposes this mismatch from the inference side by removing the direct x_t-dependent residual from the reverse update. Improved validity indicates that the denoiser's clean proposals contain useful global structure that the standard trajectory may fail to exploit. Self-correction addresses the same problem from the training side by exposing the model to its own intermediate predictions and training recovery to the original valid target.

Overall, our results suggest that continuous diffusion models can learn global constraints, but better alignment between the states used for training and the states produced during sampling remains a central challenge for reliable constrained discrete reasoning.

LLM USAGE

We used ChatGPT and Claude Code as general-purpose assistants during the preparation of this work. For writing, the authors first drafted the manuscript sections and then used ChatGPT iteratively to improve clarity, English expression, and presentation. The authors reviewed and decided on all resulting edits.

For code development, Claude Code was used to assist with implementing, debugging, and refining code based on ideas and specifications provided by the authors. Research ideas, methodological choices, experimental design, and interpretation of the results remained the responsibility of the authors. AI-assisted code and manuscript changes were reviewed and verified by the authors.

The mathematical claims and proof sketches were formulated by the authors; an LLM was then used to write out and polish the derivations, and the authors verified the resulting proofs.

ETHICS STATEMENT

This work investigates performance of continuous diffusion models on constrained discrete tasks using synthetic and publicly available benchmark datasets. It does not involve human subjects or personally identifiable data, and we do not deploy the models in high-stakes settings. Our models learn constraints implicitly from the data, so the generated outputs are not guaranteed to be valid; practitioners applying such methods in sensitive domains should verify the outputs. More broadly, as with generative models in general, downstream applications could be misused, but we do not foresee direct negative societal impacts from this work.

REPRODUCIBILITY STATEMENT

We provide a minimal repository in the supplementary material with training and evaluation code and checkpoints for the continuous models. A download script provides the publicly released SRM checkpoint used for MNIST-Sudoku and the required external assets. Samplers are described in Section 3.2 and Appendix B, and the self-correction training procedure is detailed in Appendix C. Compute resources and hyperparameters are provided in Appendix F. Architectures are described in Appendix E, and dataset details are given in Appendix D.

REFERENCES

Eloi Alonso, Adam Jelley, Vincent Micheli, Anssi Kanervisto, Amos Storkey, Tim Pearce, and François Fleuret. Diffusion for world modeling: Visual details matter in atari. *Advances in Neural Information Processing Systems*, 37:58757–58791, 2024.

Brandon Amos and J Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *International conference on machine learning*, pp. 136–145. PMLR, 2017.

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.

Pavel Avdeyev, Chenlai Shi, Yuhao Tan, Kseniia Dudnyk, and Jian Zhou. Dirichlet diffusion score model for biological sequence generation. In *International Conference on Machine Learning*, pp. 1276–1301. PMLR, 2023.

Gregor Bachmann and Vaishnavh Nagarajan. The pitfalls of next-token prediction. *arXiv preprint arXiv:2403.06963*, 2024.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. *Advances in neural information processing systems*, 28, 2015.

Michael Cardei, Jacob K Christopher, Bhavya Kailkhura, Tom Hartvigsen, and Ferdinando Fioretto. Constrained discrete diffusion. *Advances in Neural Information Processing Systems*, 38:12383–12416, 2026.

Boyuan Chen, Diego Martí Monsó, Yilun Du, Max Simchowitz, Russ Tedrake, and Vincent Sitzmann. Diffusion forcing: Next-token prediction meets full-sequence diffusion. *Advances in Neural Information Processing Systems*, 37:24081–24125, 2024.

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

Jacob K Christopher, Stephen Baek, and Ferdinando Fioretto. Constrained synthesis with projected diffusion models. *Advances in Neural Information Processing Systems*, 37:89307–89333, 2024.

Hyungjin Chung, Jeongsol Kim, Michael T Mccann, Marc L Klasky, and Jong Chul Ye. Diffusion posterior sampling for general noisy inverse problems. *arXiv preprint arXiv:2209.14687*, 2022.

Hyungjin Chung, Jeongsol Kim, Sehui Kim, and Jong Chul Ye. Parallel diffusion models of operator and image for blind inverse problems. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6059–6069. IEEE, 2023.

Yuntian Deng, Noriyuki Kojima, and Alexander M Rush. Markup-to-image diffusion models with scheduled sampling. *arXiv preprint arXiv:2210.05147*, 2022.

Honghua Dong, Jiayuan Mao, Tian Lin, Chong Wang, Lihong Li, and Denny Zhou. Neural logic machines. *arXiv preprint arXiv:1904.11694*, 2019.

Mariia Drozdova, Aidan Sirbu, Pietro Miotti, Robert Obryk, Mayalen Etcheverry, Eyvind Niklasson, and Blake Richards. Diffusion as a training curriculum for timestep-free iterative reasoning. *arXiv preprint arXiv:2609.01449*, 2026.

Yilun Du, Jiayuan Mao, and Joshua B Tenenbaum. Learning iterative reasoning through energy diffusion. *arXiv preprint arXiv:2406.11179*, 2024.

Bradley Efron. Tweedie's formula and selection bias. *Journal of the American Statistical Association*, 106(496): 1602–1614, 2011.

Alec Helbling, Andrey Bryutkin, Mauro Martino, Nima Dehmamy, and Hendrik Strobelt. Flow reasoning models: Scaling reasoning through iterative self-refinement. *arXiv preprint arXiv:2606.29150*, 2026.

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

Jonathan Ho, Tim Salimans, Alexey Gritsenko, William Chan, Mohammad Norouzi, and David J Fleet. Video diffusion models. *Advances in neural information processing systems*, 35:8633–8646, 2022.

Peter Holderrieth and Ezra Erives. An introduction to flow matching and diffusion models. *arXiv preprint arXiv:2506.02070*, 2025.

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

Xun Huang, Zhengqi Li, Guande He, Mingyuan Zhou, and Eli Shechtman. Self forcing: Bridging the train-test gap in autoregressive video diffusion. *Advances in Neural Information Processing Systems*, 38:167283–167308, 2026.

Naoto Inoue, Kotaro Kikuchi, Edgar Simo-Serra, Mayu Otani, and Kota Yamaguchi. Layoutdm: Discrete diffusion model for controllable layout generation. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10167–10176. IEEE, 2023.

Michael Janner, Yilun Du, Joshua B Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. *arXiv preprint arXiv:2205.09991*, 2022.

Alexia Jolicoeur-Martineau. Less is more: Recursive reasoning with tiny networks. *arXiv preprint arXiv:2510.04871*, 2025.

Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models. *Advances in neural information processing systems*, 35:26565–26577, 2022.

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

Chieh-Hsin Lai, Yang Song, Dongjun Kim, Yuki Mitsufuji, and Stefano Ermon. The principles of diffusion models. *arXiv preprint arXiv:2510.21890*, 2025.

Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

Tianhong Li and Kaiming He. Back to basics: Let denoising generative models denoise. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 36115–36125, 2026.

Yang Li, Lvda Chen, Haonan Wang, Runzhong Wang, and Junchi Yan. Generation as search operator for test-time scaling of diffusion-based combinatorial optimization. *Advances in Neural Information Processing Systems*, 38:127168–127196, 2026a.

Yangming Li and Mihaela van der Schaar. On error propagation of diffusion models. In *International Conference on Learning Representations*, volume 2024, pp. 32791–32807, 2024.

Zeyang Li, Kaveh Alim, and Navid Azizan. Hardflow: Hard-constrained sampling for flow-matching models via trajectory optimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2026b.

Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022.

Yaron Lipman, Marton Havasi, Peter Holderrieth, Neta Shaul, Matt Le, Brian Karrer, Ricky TQ Chen, David Lopez-Paz, Heli Ben-Hamu, and Itai Gat. Flow matching guide and code. *arXiv preprint arXiv:2412.06264*, 2024.

Tsiry Mayet, Pourya Shamsolmoali, Simon Bernard, Eric Granger, Romain Hérault, and Clement Chatelain. Td-paint: Faster diffusion inpainting through time-aware pixel conditioning. In *International Conference on Learning Representations*, volume 2025, pp. 102537–102565, 2025.

Mang Ning, Enver Sangineto, Angelo Porrello, Simone Calderara, and Rita Cucchiara. Input perturbation reduces exposure bias in diffusion models. *arXiv preprint arXiv:2301.11706*, 2023.

Mang Ning, Mingxiao Li, Jianlin Su, Albert Ali Salah, and Itir Onal Ertugrul. Elucidating the exposure bias in diffusion models. In *International Conference on Learning Representations*, volume 2024, pp. 15167–15189, 2024.

Pierre Pereira. Encoding the tsp solution on a circle. `https://pierrot-lc.dev/posts/circular-tsp/`, March 2026. URL `https://doi.org/10.5281/zenodo.20037323`.

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.

Zhiyao Ren, Yibing Zhan, Liang Ding, Gaoang Wang, Chaoyue Wang, Zhongyi Fan, and Dacheng Tao. Multi-step denoising scheduled sampling: Towards alleviating exposure bias for diffusion models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 4667–4675, 2024.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *2022 IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, pp. 10674–10685. ieee, 2022.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 627–635. JMLR Workshop and Conference Proceedings, 2011.

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

Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020a.

Yang Song and Prafulla Dhariwal. Improved techniques for training consistency models. In *International Conference on Learning Representations*, volume 2024, pp. 15078–15097, 2024.

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

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*, 2020b.

Ayhan Suleymanzade, Chanhyuk Lee, Floor Eijkelboom, Nicholas M Boffi, İsmail İlkan Ceylan, and Jinwoo Kim. Thinking with looped flows. *arXiv preprint arXiv:2609.11801*, 2026.

Zhiqing Sun and Yiming Yang. Difusco: Graph-based diffusion solvers for combinatorial optimization. *Advances in neural information processing systems*, 36:3706–3731, 2023.

Utkarsh Utkarsh, Pengfei Cai, Alan Edelman, Rafael Gomez-Bombarelli, and Christopher Rackauckas. Physics-constrained flow matching: Sampling generative models with hard constraints. *Advances in Neural Information Processing Systems*, 38:160217–160252, 2026.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Guan Wang, Jin Li, Yuhao Sun, Xing Chen, Changling Liu, Yue Wu, Meng Lu, Sen Song, and Yasin Abbasi Yadkori. Hierarchical reasoning model. *arXiv preprint arXiv:2506.21734*, 2025.

Po-Wei Wang, Priya Donti, Bryan Wilder, and Zico Kolter. Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *International Conference on Machine Learning*, pp. 6545–6554. PMLR, 2019.

Joseph L Watson, David Juergens, Nathaniel R Bennett, Brian L Trippe, Jason Yim, Helen E Eisenach, Woody Ahern, Andrew J Borst, Robert J Ragotte, Lukas F Milles, et al. De novo design of protein structure and function with rfdiffusion. *Nature*, 620(7976):1089–1100, 2023.

Christopher Wewer, Bart Pogodzinski, Bernt Schiele, and Jan Eric Lenssen. Spatial reasoning with denoising models. *arXiv preprint arXiv:2502.21075*, 2025.

Zhen Xing, Qijun Feng, Haoran Chen, Qi Dai, Han Hu, Hang Xu, Zuxuan Wu, and Yu-Gang Jiang. A survey on video diffusion models. *ACM Computing Surveys*, 57(2):1–42, 2024.

Jiacheng Ye, Jiahui Gao, Shansan Gong, Lin Zheng, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Beyond autoregression: Discrete diffusion for complex reasoning and planning. In *International Conference on Learning Representations*, volume 2025, pp. 77875–77898, 2025.

Junyu Zhang, Daochang Liu, Eunbyung Park, Shichao Zhang, and Chang Xu. Anti-exposure bias in diffusion models. In *International Conference on Learning Representations*, volume 2025, pp. 77586–77606, 2025.

A Diffusion and parameterizations details

This appendix expands the diffusion notation, parameterization conversions, and continuous-time derivations summarized in Sec. 2.

A.1 Forward Markov chain

The forward marginal used in the main text is

$$x_t = \alpha(t)x_0 + \beta(t)\epsilon, \qquad \epsilon \sim \mathcal{N}(0, I), \qquad \beta(t) = \sqrt{1-\alpha(t)^2}.$$

It induces

$$q(x_t \mid x_0) = \mathcal{N}\left(\alpha(t)x_0, (1-\alpha(t)^2)I\right). \tag{10}$$

Equivalently, the same marginals can be generated by the discrete Markov chain

$$x_t = \sqrt{\delta(t)}\, x_{t-1} + \sqrt{1-\delta(t)}\, \epsilon_t, \qquad \epsilon_t \sim \mathcal{N}(0, I), \tag{11}$$

with transition kernel

$$q(x_t \mid x_{t-1}) = \mathcal{N}\left(\sqrt{\delta(t)}x_{t-1}, (1-\delta(t))I\right).$$

The one-step coefficient is chosen so that equation 11 reproduces the marginals in equation 10:

$$\delta(t) = \frac{\alpha(t)^2}{\alpha(t-1)^2}, \qquad \alpha(t)^2 = \prod_{s=1}^{t} \delta(s). \tag{12}$$

A.2 DDPM posterior

Since the forward process is Gaussian, the posterior conditional on the clean sample is tractable (Ho et al., 2020):

$$q(x_{t-1} \mid x_t, x_0) = \mathcal{N}\left(\mu_t(x_t, x_0), \tilde{\beta}_t I\right), \tag{13}$$

with mean and variance given by:

$$\mu_t(x_t, x_0) = \alpha(t-1)x_0 \frac{1-\delta(t)}{1-\alpha(t)^2} + \frac{\sqrt{\delta(t)}\left(1-\alpha(t-1)^2\right)}{1-\alpha(t)^2} x_t, \tag{14}$$

$$\tilde{\beta}_t = \left(1-\alpha(t-1)^2\right)\frac{1-\delta(t)}{1-\alpha(t)^2}. \tag{15}$$

Thus DDPM sampling replaces the unknown x_0 with the model prediction $\hat{x}_0(x_t, t)$ and samples from

$$p_\theta(x_{t-1} \mid x_t) \doteq q\left(x_{t-1} \mid x_t, \hat{x}_0(x_t, t)\right).$$

The posterior mean can also be written in an anchored form:

$$\mu_t(x_t, x_0) = \alpha(t-1)x_0 + B_t\left(x_t - \alpha(t)x_0\right), \qquad B_t \doteq \frac{\sqrt{\delta(t)}\left(1-\alpha(t-1)^2\right)}{1-\alpha(t)^2}. \tag{16}$$

This decomposition separates the clean-sample component from the residual inherited from the current noisy state. In the main paper, this residual is the source of the anchoring effect that distinguishes DDPM-style updates from pure Tweedie reprojection.

A.3 Prediction parameterizations

A diffusion model can be parameterized to predict the clean sample x_0, the noise ϵ (Ho et al., 2020), the marginal score $s_t(x_t) = \nabla_{x_t} \log q_t(x_t)$ (Song & Ermon, 2019), or the marginal velocity $u_t(x_t)$ in the flow-matching formulation (Lipman et al., 2022; 2024). The prediction parameterization and the loss target are separate choices (Li & He, 2026). At the population optimum, these quantities are equivalent up to deterministic conversions induced by the forward process (Lai et al., 2025).

Let

$$x_0^*(x_t, t) \doteq \mathbb{E}[x_0 \mid x_t].$$

Then the corresponding optimal noise, score, and velocity predictions are

$$\epsilon^*(x_t,t) = \frac{x_t - \alpha(t)x_0^*(x_t,t)}{\beta(t)}, \qquad s^*(x_t,t) \doteq \nabla_{x_t}\log q_t(x_t) = -\frac{\epsilon^*(x_t,t)}{\beta(t)},$$
$$u^*(x_t,t) = \frac{\dot\beta(t)}{\beta(t)}x_t + \left(\dot\alpha(t) - \frac{\alpha(t)\dot\beta(t)}{\beta(t)}\right)x_0^*(x_t,t). \tag{17}$$

Tweedie's formula gives

$$x_0^*(x_t,t) = \mathbb{E}[x_0 \mid x_t] = \frac{x_t + \beta(t)^2\nabla_{x_t}\log q_t(x_t)}{\alpha(t)}.$$

Therefore an x-prediction model can be used in score-, noise-, or velocity-based samplers by applying equation 17. In our experiments, we use x-prediction and the x-prediction loss in equation 4, following the practical distinction between prediction type and loss target highlighted by Li & He (2026).

A.4 Continuous-time derivation

For continuous-time samplers, we view the variance-preserving forward process as a probability path $\{q_t\}_{t\in[0,T]}$:

$$x_t = \alpha(t)x_0 + \beta(t)\epsilon, \qquad q_t(x \mid x_0) = \mathcal{N}\left(\alpha(t)x_0, \beta(t)^2 I\right), \qquad \epsilon \sim \mathcal{N}(0,I). \tag{18}$$

Here t is continuous, $\alpha(0)=1$, $\alpha(T)\approx 0$, and $\beta(t) = \sqrt{1-\alpha(t)^2}$.

Conditional and marginal vector fields. Differentiating equation 18 with respect to t, for a fixed pair (x_0,ϵ), gives the conditional velocity field

$$u_t(x \mid x_0) = \dot\alpha(t)x_0 + \dot\beta(t)\epsilon = \frac{\dot\beta(t)}{\beta(t)}x + \left(\dot\alpha(t) - \alpha(t)\frac{\dot\beta(t)}{\beta(t)}\right)x_0, \tag{19}$$

where we used

$$\epsilon = \frac{x - \alpha(t)x_0}{\beta(t)}.$$

Since $u_t(x \mid x_0)$ is linear in x_0, the marginal vector field is obtained by replacing x_0 with $\mathbb{E}[x_0 \mid x_t = x]$. Using Tweedie's formula gives

$$u_t(x) = \frac{\dot\alpha(t)}{\alpha(t)}x + \left(\beta(t)^2\frac{\dot\alpha(t)}{\alpha(t)} - \beta(t)\dot\beta(t)\right)\nabla_x\log q_t(x). \tag{20}$$

Probability-flow ODE and SDE family. The deterministic dynamics with marginals q_t are given by the probability-flow ODE

$$\mathrm{d}x = u_t(x)\,\mathrm{d}t. \tag{21}$$

More generally, for any non-negative diffusion schedule $\sigma(t)$, the reverse-time SDE driven by a standard Wiener process w_t,

$$\mathrm{d}x = \left[u_t(x) - \frac{\sigma(t)^2}{2}\nabla_x\log q_t(x)\right]\mathrm{d}t + \sigma(t)\,\mathrm{d}w_t$$

has the same marginals q_t in the exact-score and exact-solver limit (Holderrieth & Erives, 2025, Thm. 17). The probability-flow ODE is the special case $\sigma(t)=0$. The variance-preserving reverse SDE of Song et al. (2020b) corresponds to

$$\sigma(t)^2 = -2\frac{\dot\alpha(t)}{\alpha(t)}.$$

Numerical samplers. The continuous-time formulation allows sampling by numerically solving either the probability-flow ODE or the SDE family. We use ODE solvers such as Euler and Heun, and SDE solvers such as Euler–Maruyama (Song et al., 2020b; Karras et al., 2022). In all cases, we plug the network prediction $\hat{x}_0(x_t,t)$ into the required score or velocity expression using the conversions in equation 17. DDPM and DDIM can also be derived from this continuous-time perspective (Song et al., 2020a;b).

B SAMPLERS

We explain various sampling update rules below and then describe the overall sampling process for both generation and completion/in-painting with the pinning procedure.

B.1 DDIM GENERALIZED FORMULA: DDIM, DDPM AND TWEEDIE REPROJECTION

DDIM (Song et al., 2020a) introduces a family of forward processes with the same marginal distributions in equation 1 as DDPM (Ho et al., 2020). Their generalized sampling update takes the form:

$$q_\kappa(x_{t-1} \mid x_t, \hat{x}_0) = \mathcal{N}\big(\alpha(t-1)\hat{x}_0 + \sqrt{1-\alpha(t-1)^2-\kappa_t^2}\cdot\frac{x_t-\alpha(t)\hat{x}_0}{\sqrt{1-\alpha(t)^2}}, \kappa_t^2 I\big) \tag{22}$$

$$= \mathcal{N}\big(\alpha(t-1)\hat{x}_0 + \sqrt{1-\alpha(t-1)^2-\kappa_t^2}\cdot\hat{\epsilon}_t, \kappa_t^2 I\big) \tag{23}$$

Different choices of κ_t^2 recover known samplers: $\kappa_t^2 = \tilde{\beta}_t = (1-\alpha(t-1)^2)\cdot\frac{1-\delta(t)}{1-\alpha(t)^2}$ yields DDPM (Ho et al., 2020), while $\kappa_t^2 = 0$ gives the deterministic DDIM update (Song et al., 2020a).

Our Tweedie reprojection corresponds to $\kappa_t^2 = 1-\alpha(t-1)^2$. This endpoint relation does not imply exact marginal preservation after replacing the true clean sample x_0 by the learned prediction $\hat{x}_0(x_t, t)$, even if that prediction equals the exact conditional mean.

Linear-path sampling for SRM. For the linear path $x_t = (1-t)x_0 + t\epsilon$, with normalized time $t \in [0,1]$, write $v_\theta(x_t, t)$ for the corresponding velocity prediction. The clean and noise estimates are

$$\hat{x}_0 = x_t - t v_\theta(x_t, t), \qquad \hat{\epsilon} = x_t + (1-t)v_\theta(x_t, t).$$

The deterministic generalized DDIM update to $s < t$ is therefore

$$x_s = (1-s)\hat{x}_0 + s\hat{\epsilon} = x_t + (s-t)v_\theta(x_t, t),$$

which is an Euler step on the linear path. Tweedie reprojection instead uses $x_s = (1-s)\hat{x}_0 + s\epsilon'$, with fresh $\epsilon' \sim \mathcal{N}(0, I)$.

B.2 TIME DISCRETIZATIONS

Whereas we use uniform / linear time discretization (equidistant timesteps) throughout our work, others can also be used in practice (Karras et al., 2022).

We will denote a discretization of $N \in \mathbb{N}^+$ time points with:

$$\{t_{N-1}, t_{N-2}, \ldots, t_0\}, \qquad t_{N-1} = T, t_0 = 0. \tag{24}$$

where we also use $h_i \doteq t_i - t_{i-1}$ as positive step sizes since timesteps are decreasing.

To compare the generalized sampling update in equation 22 (including DDIM, DDPM, and Tweedie reprojection) with other samplers, we can replace it as follows:

$$x_s = \alpha(s)\hat{x}_0 + \sqrt{1-\alpha(s)^2-\kappa_{t_i}^2}\cdot\frac{x_{t_i}-\alpha(t_i)\hat{x}_0}{\sqrt{1-\alpha(t_i)^2}} + \kappa_{t_i}\cdot\epsilon(t_i) \tag{25}$$

with $\epsilon(t_i) \sim \mathcal{N}(0, I)$ and $i = N-1, \ldots, 1$.

B.3 NUMERICAL ODE AND SDE SOLVERS: EULER, HEUN, AND EULER-MARUYAMA

The Euler and Heun (aka improved Euler) methods are first- and second-order ODE solvers, popular for their low Number of Function Evaluations (NFE) and ease of implementation.

They become deterministic samplers once used to solve the PF-ODE in equation 21. Starting from $x_T \sim \mathcal{N}(0, I)$, Euler and Heun iterate from $i = N-1$ to $i = 1$, using the estimate of the marginal vector field, retrieved from the model's x-prediction.

Euler update rule

$$x_s = x_{t_i} - h_i \cdot \hat{u}_{t_i}(x_{t_i}), \tag{26}$$

Euler-Maruyama (EM) update rule Recall that for *any non-negative* diffusion schedule σ_t, the stochastic dynamics

$$\mathrm{d}x = \left[u_t(x) - \tfrac{\sigma_t^2}{2}\nabla_x \log q_t(x)\right]\mathrm{d}t + \sigma_t \mathrm{d}w. \tag{27}$$

share the same marginals q_t (Holderrieth & Erives, 2025, Thm. 17), reducing to the deterministic case in equation 21 for $\sigma_t = 0$.

Euler-Maruyama (EM) is a numerical SDE solver that can be used as a stochastic sampler. Starting from $x_T \sim \mathcal{N}(0, I)$, EM iterates from $i = N-1$ to $i = 1$, using the estimate of the marginal vector field and score, retrieved from the model's x-prediction.

$$x_s = x_{t_i} + h_i \cdot \hat{f}(x_{t_i}, t_i) + g(t_i)\sqrt{h_i} \cdot \epsilon(t_i), \qquad \epsilon(t_i) \sim \mathcal{N}(0, I) \tag{28}$$

where we denote $\hat{f}(x,t) \doteq -\hat{u}_t(x) + \frac{\sigma_t^2}{2}\hat{s}_t(x)$ as the *drift* coefficient and $g(t) \doteq \sigma_t$ as the *diffusion* coefficient.

In our work, we refer to EM as using a fixed $g(t_i) = \sigma$, and EM decay as using a *decreasing* diffusion schedule σ_t.

Euler–Maruyama noise scale. The SDE family in Eq. 6 motivates treating $\sigma(t)$ as a sampler hyperparameter. Theoretically, changing $\sigma(t)$ changes the stochastic dynamics but not the marginal path q_t, assuming exact scores and exact integration. In practice, finite-step solvers and learned denoisers introduce discretization and model error, so the choice of $\sigma(t)$ affects empirical performance. We therefore tune the EM noise scale by five-fold cross-validation. For EM decay, we also tune the time at which $\sigma(t)$ begins linearly annealing to zero near the data endpoint.

B.4 Sampling process and pinning procedure

The algorithm 1 summarizes the overall sampling process for both unconditional and conditional generation (completion or in-painting), with pinning procedures for the latter. Recall that we go from x_T to x_0 when sampling.

Algorithm 1: (Un-)conditional generation. Inputs are the time discretization, optional conditioning c, its corresponding mask m

Procedure `sample` $(\{t_{N-1}, t_{N-2}, \ldots, t_0\}, c, m)$ **:**
- Sample initial tensor $x_T \sim \mathcal{N}(0, I)$
- Initialize buffer to store trajectory $\{x_{t_i}\}_i$
- **for** $i = N-1$ **to** 1 **do**
  - $s = t_{i-1}$
  - Compute step size $h_i = t_i - s$
  - Sample $x_s =$ `update_rule` $(x_{t_i}, t_i, h_i, \ldots)$
  - **if** $m \neq \emptyset$ **then**
    - `pin` (x_s, s, c, m)
  - Append x_s to buffer

C Self-correction algorithm

We summarize the SELF-CORRECTION training procedure in Algorithm 2. We denote $\mathrm{sg}(.)$ as the stop-gradient operator.

Algorithm 2: SELF-CORRECTION training for (un-)conditional generation on a batch of size B. Inputs are the regularization coefficient λ_{simple}, optional conditioning c and its corresponding mask m. $\text{sg}(\cdot)$ denotes the stop-gradient operator; every MSE is averaged over non-conditioned cells only.

Procedure `training_procedure(`λ_{simple}, c, m`)`**:**
  for *steps* **do**
    `update_diffusion_model(`λ_{simple}, c, m`)`

Procedure `update_diffusion_model(`λ_{simple}, c, m`)`**:**
  Sample a batch $x_0 \sim q_0(x)$ of size B
  Sample $t_1 \sim \mathcal{U}[0, T]$ and $t_2 \sim \mathcal{U}[0, t_1]$
  Sample $x_{t_1} \sim q(x_{t_1} \mid x_0)$
  if $m \neq \emptyset$ **then**
    `pin(`x_{t_1}, c, m`)`
  Compute $\hat{x}_0 = \text{sg}\big(f_\theta(x_{t_1}, t_1)\big)$
  if $m \neq \emptyset$ **then**
    `pin(`$\hat{x}_0, c, m$`)`
  Sample $\tilde{x}_{t_2} \sim q(x_{t_2} \mid \hat{x}_0)$
  if $m \neq \emptyset$ **then**
    `pin(`$\tilde{x}_{t_2}, c, m$`)`
  Compute $\hat{x}'_0 = f_\theta(\tilde{x}_{t_2}, t_2)$
  Compute recovery loss $\mathcal{L}_{\text{rec}}(\theta) = \text{MSE}(\hat{x}'_0, x_0)$
  Take a sub-batch $\mathcal{B}_s$ of $\lfloor \lambda_{\text{simple}} B \rfloor$ samples of x_0
  Sample $t_3 \sim \mathcal{U}[0, T]$ and $x_{t_3} \sim q(x_{t_3} \mid x_0)$ on $\mathcal{B}_s$
  if $m \neq \emptyset$ **then**
    `pin(`x_{t_3}, c, m`)`
  Compute simple loss $\mathcal{L}_{\text{simple}}(\theta) = \text{MSE}\big(f_\theta(x_{t_3}, t_3), x_0\big)$ on $\mathcal{B}_s$
  Compute final loss $\mathcal{L}_{\text{SC}}(\theta) = \mathcal{L}_{\text{rec}}(\theta) + \lambda_{\text{simple}} \mathcal{L}_{\text{simple}}(\theta)$
  Update f_θ with one backward pass on $\mathcal{L}_{\text{SC}}(\theta)$

This procedure generalizes naturally to multiple prediction steps. Longer unrolls were unstable when trained from scratch; the warm-started multi-step experiment in Appendix I trained successfully but did not outperform one-step self-correction.

Computational overhead of self-correction training. Standard denoising training performs one gradient forward pass and one backward pass per optimizer step. Self-correction training adds (i) one additional *gradient-free* forward pass, which produces the model prediction that is re-noised to the second time point, and (ii) one gradient forward pass on a sub-batch of $\lfloor \lambda_{\text{simple}} B \rfloor$ samples for $\mathcal{L}_{\text{simple}}$; a single backward pass is taken on the combined objective. Counting a backward pass as twice the cost of a forward pass, this predicts a per-step training cost of $(4 + 3\lambda_{\text{simple}})/3 \approx 1.43\times$ the baseline for $\lambda_{\text{simple}} = 0.1$. Both variants are trained for the same number of optimizer steps (2×10^6), and inference cost is unchanged, as self-correction modifies only the training objective and not the sampler.

D ADDITIONAL TASKS DETAILS

We train and evaluate on the following benchmarks for constrained data generation and in-painting/completion with a focus on discrete-space reasoning tasks: Sudoku-Extreme (Wang et al., 2025; Jolicoeur-Martineau, 2025), MNIST Sudoku (Wewer et al., 2025), graph-connectivity (GC, Du et al., 2024) and two datasets we introduce: Latin square and N-queens. Throughout, clues denote randomly positioned revealed cells.

Sudoku. We consider two separate Sudoku training regimes, both based on the Sudoku-Extreme dataset (Wang et al., 2025; Jolicoeur-Martineau, 2025), which provides partial 9×9 grids together with their completions. For the *Sudoku* setting, we train on completed boards from the training

Table 2: **Dataset summary.** Throughout, clues denote partial observations the model conditions on: randomly positioned revealed cells with count $k \sim \mathcal{U}[\![a, b]\!]$ where $[\![a, b]\!] = \{a, \dots, b\}$ (Sudoku, Latin square, N-queens); or the full adjacency matrix (GC, always given at train and inference). † Out-of-distribution.

	Sudoku	Sudoku-Extreme	GC	Latin square	N-queens
Grid size	9×9	9×9	$N \times N$	7×7	14×14
Cells m	81	81	N^2	49	14
Vocab d	9	9	2	7	14
Train clues	$[\![0, 80]\!]$	$[\![17, 35]\!]$	Adj. mat. $N = 12$	$[\![0, N^2 - 1]\!]$	$[\![0, N - 1]\!]$
Eval clues	21 $[\![27, 53]\!]$ (Med.) $[\![0, 26]\!]$ (Hard)	$[\![17, 36]\!]$	Adj. mat., $N = 12$ Adj. mat., $N = 18^\dagger$	random: 14 generation: 0	random: 7 generation: 0
Validity	Sudoku rules	Sudoku rules	Connectivity matches ground truth	One symbol per row/col	No two queens attack

split and generate conditioning masks on the fly, with the number of clues sampled uniformly from $\{0, \dots, 80\}$. We evaluate this model with 21 clues, as well as the Medium and Hard clue-count settings of SRM (Wewer et al., 2025), corresponding to clue counts sampled from $\{27, \dots, 53\}$ and $\{0, \dots, 26\}$, respectively. For the *Sudoku-Extreme* setting, we train a separate model using the dataset-provided partial grids from the Sudoku-Extreme training split and evaluate it on the corresponding test split.

For evaluation, we sample 2000 puzzles per seed from the corresponding test split, using three seeds. For the Sudoku clue-count settings, masks are generated on the fly according to the evaluation regime, whereas for Sudoku-Extreme we use the dataset-provided partial grids. Since the Sudoku-Extreme test set is large, we evaluate on a uniformly sampled subset and report binomial confidence intervals; the standard error is at most 1.12 percentage points, corresponding to a 95% confidence interval of approximately ± 2.19 percentage points.

Graph connectivity. Graph connectivity (Du et al., 2024) is a dataset for graphs with N nodes, consisting of both $N \times N$ binary adjacency and $N \times N$ binary connectivity matrices. The latter describes the existence of a path between nodes. Conditioned on adjacency matrices, we train and evaluate models to generate connectivity matrices. We define a connectivity matrix as valid if it matches the true connectivity matrix. We do not use any random number of clues, and following prior work (Du et al., 2024), we only train with at most $N = 12$ nodes, and evaluate on $N = 12$ and $N = 18$. For training and evaluation, we use their train and test dataset generators (seeds are different), which continually provide samples, meaning that there is no fixed dataset size.

MNIST Sudoku. We use the MNIST-Sudoku dataset of Wewer et al. (2025), where each 9×9 puzzle is rendered as a 252×252 grayscale image. Every cell contains a 28×28 MNIST (LeCun, 1998) digit image sampled from a random instance of the corresponding digit class, requiring the model to jointly perform digit recognition and Sudoku constraint reasoning directly from raw pixels. Following the original split convention, we reserve the last 1,000 puzzles for evaluation. Inputs are scaled to $[-1, 1]$ and treated as continuous-valued images without tokenization; the diffusion model predicts the full 252×252 board. This experiment is separate from the mini MNIST-Sudoku models trained with 8×8 digits in Appendix K.

Latin square. Latin squares of size $N \times N$ are arrays with values taken among N different symbols. Arrays are valid if each symbol occurs exactly once per row and once per column. In our case, we use $N = 7$ and randomized backtracking to create 210000 valid Latin squares, where 80% is for the training set. We train with a number of clues uniformly drawn from $\{0, \dots, N^2 - 1\}$ and evaluate in random in-painting and unconditional generation.

N-Queens. An N-Queens board contains N queens, with no two sharing a row, column, or diagonal. We represent each board as a vector in $\{1, \dots, N\}^N$, whose i-th entry gives the queen's column in row i. We use $N = 14$ and construct the dataset by sampling uniformly from the complete

set of $365{,}596$ valid boards. The training split contains $168{,}000$ distinct boards. During training, the number of revealed queens is sampled uniformly from $\{0, \ldots, N-1\}$. We evaluate both completion from randomly revealed queens and unconditional generation.

D.1 Continuous representations from discrete configurations

From discrete configurations of m cells, we convert each token into their one-hot representation (Chen et al., 2022), giving $m \times d$ tensors where the last dimension is the one-hot dimension. Please refer to Table 2 for their values.

E Model architectures and hyperparameters

E.1 Discrete diffusion

We additionally train absorbing-state discrete-diffusion models (Austin et al., 2021) as external references for Table 1. For Sudoku, N-Queens, and Latin squares, we use a four-layer Transformer of width 128, approximately matching the size of our continuous model. For graph connectivity, we use the same NLM backbone, replacing the reachability input by a categorical $0/1/$`[MASK]` representation. Training randomly masks target tokens and minimizes clean-token cross-entropy at masked positions; conditioning clues (or the graph adjacency matrix) remain fixed.

We evaluate three inference procedures: ancestral unmasking, confidence-based unmasking, and remasking, which adds four refinement sweeps that re-predict the lowest-confidence 20% of non-clue tokens. We use 256 sampling steps for Sudoku, N-Queens, and Latin squares, $1{,}000$ for Sudoku-Extreme, and 64 for graph connectivity. Checkpoints are selected on held-out validation data and evaluated on test data.

E.2 Architecture for Sudoku, N-Queens, and Latin

For all tasks except MNIST Sudoku (Wewer et al., 2025) and Graph Connectivity (GC) (Du et al., 2024) (see Appendix E.3), our network f_θ is a Transformer (Vaswani et al., 2017), taking as input a continuous sample x_t and diffusion time t:

- The sample x_t is linearly embedded in $\mathbb{R}^{d_{\text{emb}}}$ and added to its learnable positional embeddings. For Sudoku, we use three additive learnable positional embeddings (row, column, and block) instead of one.
- The diffusion time t is embedded using Fourier features $[\sin(2\pi\omega_k \frac{t}{T}), \cos(2\pi\omega_k \frac{t}{T})]_{k=1}^{d_{\text{time}}/2}$ with log-spaced frequencies ω_k, followed by a two-layer MLP with SiLU activations mapping the embedding into $\mathbb{R}^{d_{\text{emb}}}$.
- We then add them together and feed them through a sequence of L Transformer blocks, followed by a layer normalization and linear layer.

We use pre-norm Transformer blocks (Radford et al., 2019) with *full* self-attention (Vaswani et al., 2017) and per-position MLP sublayers, each wrapped with residual connections. Within each Transformer block, we apply dropout after each attention and MLP sublayer, as well as attention weight dropout inside the multi-head attention.

Except for the number of continuous tokens (i.e., number of cells m) and vocabulary size (i.e., one-hot dimension d) (see Table 2), all hyper-parameters are shared across tasks described in this subsection. Please refer to Table 3 for their values and to Algorithm 2 for the training procedure.

E.3 Architectures for Graph Connectivity and MNIST Sudoku

For Graph Connectivity, we adopt the Neural Logic Machine-based (Dong et al., 2019) architecture of IRED (Du et al., 2024, Table 11) and use the same hyperparameters as them.

As for MNIST Sudoku, we use the pre-trained SRM model (Wewer et al., 2025) without any additional training, and therefore have no architecture hyperparameters to report.

Table 3: Architecture details

Name	Short-hand	Value
Embedding dimension	d_{emb}	128
Time embedding dimension	d_{time}	64
Time embedding MLP	–	$[d_{\text{time}}, d_{\text{emb}}, d_{\text{emb}}]$
Depth	L	4
Attention heads	n_{heads}	8
Dropout probability	p_{drop}	0.01
MLP hidden dimension	d_{mlp}	$4 \times d_{\text{emb}} = 512$, GeLU

F HYPER-PARAMETERS AND HARDWARE

Table 4: **Training and evaluation hyperparameters.** All main VP-continuous diffusion tasks share optimizer type, learning rate, and step budget; batch size is the only task-specific training knob, and sampler step budgets differ only for Sudoku-Extreme and SRM.

Hyperparameter	Value
Training	
Training loop	
Optimizer steps	2×10^6
Batch size (default)	256
Batch size (GC)	64
Mixed precision	AMP, float16
Hardware	NVIDIA RTX 3090
Optimization	
Optimizer	Adam
Learning rate	1×10^{-3}
Diffusion	
Model parameterization	x-prediction
Loss target	x-loss
Noise schedules	VP-cosine
Self-correction loss	
λ_{simple} (default)	0.1
Evaluation	
Diffusion sampling	
Sampler steps (default)	200
Sampler steps (Sudoku-Extreme)	1000
Sampler steps (SRM)	500

F.1 EXPERIMENTS COMPUTE RESOURCES

Training was done on a single RTX 3090 with 24 GiB of VRAM with a single worker. Each training run took approximately 17 hours on a Sudoku task, 44 hours on a GC task, 14 hours on a Latin task, 10 hours on a N-Queens task.

Standard paper evaluations complete within approximately 5–15 minutes per checkpoint across all samplers and regimes. Cross-validation experiments for graph connectivity are more computationally intensive, usually requiring around 12–25 minutes for 5-fold evaluation with EM and EM decay samplers. The most expensive setting is Sudoku Extreme pass@10 evaluation, which can take approximately 30–50 minutes per sampler due to the large number of samples and denoising steps.

Table 5: **Best settings in the averaged EM-decay validation sweeps.** Scores are averaged across training seeds and validation folds. These are validation diagnostics, not test results.

Regime	Training	σ	τ_{start}	Validation validity
Sudoku-Extreme	Baseline	14	0.8	0.177
Sudoku-Extreme	Self-correction	20	0.7	0.531
Sudoku 21 clues	Baseline	7	0.7	0.843
Sudoku 21 clues	Self-correction	7	0.7	0.982

F.2 Sensitivity of EM decay to sampler hyperparameters

We examine the dependence of EM decay on its initial noise scale σ and the reverse-progress value τ_{start} at which the noise begins to decay, with $\tau = 0$ at noise and $\tau = 1$ at data. The following values are validation results averaged across training seeds and validation folds.

On Sudoku-Extreme, the baseline model achieves 0.177 validity with $(\sigma, \tau_{\text{start}}) = (14, 0.8)$, but only 0.010 when σ is increased to 20 at the same decay time. For the self-corrected model, validity is 0.531 at $(20, 0.7)$, but falls to 0.138 when decay begins at 0.8 instead. Thus, increasing the noise scale or delaying its decay does not consistently improve validity. EM decay requires careful validation-based selection of these parameters, whereas Tweedie reprojection has neither of these two sampler parameters.

G Existing assets and licenses

We use existing benchmark datasets, checkpoints, and reference implementations only for research evaluation. Table 6 summarizes the license information available to us. When a dataset license is not specified separately, we report the license of the associated code or release.

Table 6: Existing assets used in this work.

Asset	License / terms
HRM / Sudoku-Extreme	Apache-2.0 code release; datasets/checkpoints via HRM
IRED / Graph Connectivity	MIT code release
SRM / MNIST-Sudoku	MIT code release; datasets/checkpoints via SRM

The assets come from HRM (Wang et al., 2025), IRED (Du et al., 2024), and SRM (Wewer et al., 2025).

H Mechanistic Analysis of DDPM-Tweedie Sampler Gap

This appendix gives a mechanistic explanation of the DDPM-Tweedie gap. Decoded validity depends on cellwise argmax decisions: Gaussian perturbations preserve a decoded state when the sampler center has sufficient margin relative to the noise scale. We use this margin view to compare the centers of Tweedie reprojection and DDPM ancestral sampling. Tweedie recenters the next state around the denoiser's prediction, while DDPM also retains a residual component from the current noisy state. We show that this residual turns out to be harmful for discrete constrained problems.

H.1 Setup and decoded stability

We use the convention 0 for data and T for noise. Let

$$\alpha(t) = \cos\left(\frac{\pi t}{2T}\right), \qquad \beta(t) = \sin\left(\frac{\pi t}{2T}\right), \qquad \alpha(t)^2 + \beta(t)^2 = 1.$$

We write

$$\hat{x}_{0,t} = f_\theta(x_t, t).$$

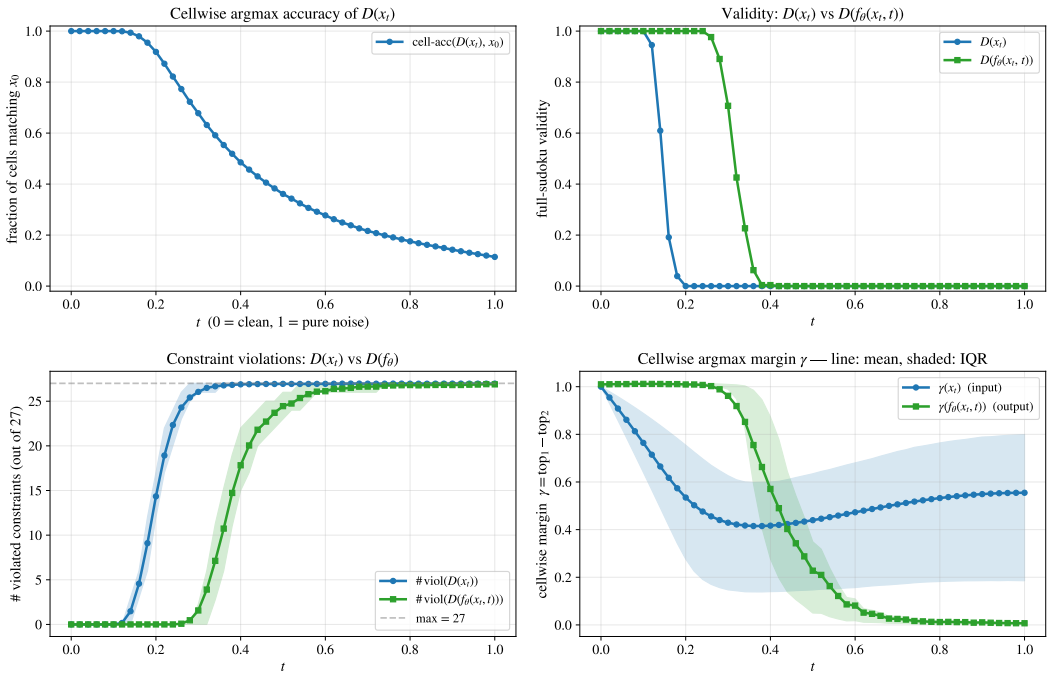


Figure 3: **Forward-noise diagnostics for the baseline model.** We evaluate $N = 64$ Sudoku puzzles with $n_{\text{rep}} = 4$ noise draws. Panels show: cellwise argmax accuracy of $D(x_t)$, full-grid validity of $D(x_t)$ and $D(f_\theta(x_t, t))$, number of violated Sudoku constraints, and argmax margin $\gamma = \text{top}_1 - \text{top}_2$. Lines are means; shaded bands show IQR across puzzles and noise draws.

For one-hot tasks, let

$$D : \mathbb{R}^{m \times d} \to \{1, \dots, d\}^m$$

be the cellwise argmax decoder, where m is the number of active/free cells and d is the number of symbols per cell. For Sudoku with 21 clues, $m = 60$; for a 28-clue subset, $m = 53$. Let

$$\mathcal{V} \subseteq \{1, \dots, d\}^m$$

be the set of valid decoded configurations and

$$\mathcal{I} = \{1, \dots, d\}^m \setminus \mathcal{V}$$

be the set of invalid configurations.

For $x \in \mathbb{R}^{m \times d}$, define

$$\text{gap}(x) = \min_i \left[x_{i, D_i(x)} - \max_{a \neq D_i(x)} x_{i,a} \right].$$

H.2 One-hot representations corrupted with Gaussian noise

For one-hot encoded tasks, the decoded output

$$x_t = c_t + \nu_t \epsilon$$

is stable when the Gaussian center c_t has a sufficiently large cellwise argmax margin relative to the Gaussian noise scale ν_t.

Figure 3 shows this effect for forward-noised one-hot Sudoku solutions

$$x_t = \alpha(t) x_0 + \beta(t) \epsilon.$$

Cellwise accuracy of the raw decoder $D(x_t)$ decreases gradually with t (top left), but full-grid validity collapses much earlier (top right): a single flipped cell is enough to invalidate the decoded Sudoku.

The trained denoiser extends the useful noise range. Its decoded output $D(f_\theta(x_t, t))$ (top right) remains valid for larger t, has fewer constraint violations (bottom left), and maintains a larger argmax margin than the raw noisy input (bottom right).

For any Gaussian sampler the following is true:

Lemma H.1 (Argmax stability)**.** *Let $c \in \mathbb{R}^{m\times d}$ satisfy*

$$\mathrm{gap}(c) \geq \gamma > 0,$$

and let

$$\epsilon \sim \mathcal{N}(0, \nu^2 I_{md}).$$

Then

$$\mathbb{P}(D(c+\epsilon) \neq D(c)) \leq m(d-1)\Phi\left(-\frac{\gamma}{\sqrt{2}\,\nu}\right),$$

where Φ is the standard Gaussian CDF.

Proof. Fix a cell i, let $j = D_i(c)$, and consider a competitor $a \neq j$. Since

$$\mathrm{gap}(c) \geq \gamma,$$

we have

$$c_{i,j} - c_{i,a} \geq \gamma.$$

The competitor overtakes j after adding noise only if

$$c_{i,a} + \epsilon_{i,a} \geq c_{i,j} + \epsilon_{i,j},$$

or equivalently

$$\epsilon_{i,a} - \epsilon_{i,j} \geq c_{i,j} - c_{i,a} \geq \gamma.$$

Since

$$\epsilon_{i,a} - \epsilon_{i,j} \sim \mathcal{N}(0, 2\nu^2),$$

this event has probability at most

$$\Phi\left(-\frac{\gamma}{\sqrt{2}\,\nu}\right).$$

A union bound over all $m(d-1)$ competitors gives the claim. □

The lemma formalizes the idea that a continuous point with large cellwise margin decodes stably under small Gaussian perturbations.

H.3 DDPM EQUALS TWEEDIE PLUS STATE MEMORY

Proposition H.2 (DDPM and Tweedie-reprojection relation)**.** *At step t, the DDPM ancestral Gaussian kernel has center*

$$c_t^{\mathrm{DDPM}} = c_t^{\mathrm{Tw}} + B_t\big(x_t - \alpha(t)\hat{x}_{0,t}\big), \qquad c_t^{\mathrm{Tw}} = \alpha(t-1)\hat{x}_{0,t},$$

where

$$B_t = \frac{\sqrt{\delta(t)}\,\big(1-\alpha(t-1)^2\big)}{1-\alpha(t)^2}, \qquad \alpha(t) = \sqrt{\delta(t)}\alpha(t-1).$$

Thus, relative to Tweedie, DDPM adds a state-anchor residual. The associated noise scales are

$$\nu_t^{\mathrm{Tw}} = \beta(t-1), \qquad \nu_t^{\mathrm{DDPM}} = \tau_t, \qquad \tau_t^2 = \frac{(1-\delta(t))\,\big(1-\alpha(t-1)^2\big)}{1-\alpha(t)^2}.$$

Proof. Following the equation 14, the DDPM ancestral mean can be written as

$$\mu_t^{\mathrm{DDPM}} = A_t\hat{x}_{0,t} + B_t x_t,$$

with

$$A_t = \alpha(t-1)\frac{1-\delta(t)}{1-\alpha(t)^2}, \qquad B_t = \frac{\sqrt{\delta(t)}\,\big(1-\alpha(t-1)^2\big)}{1-\alpha(t)^2}.$$

Using

$$\alpha(t) = \sqrt{\delta(t)}\alpha(t-1),$$

we have

$$A_t + B_t\alpha(t) = \alpha(t-1)\frac{1-\delta(t)+\delta(t)\left(1-\alpha(t-1)^2\right)}{1-\alpha(t)^2} = \alpha(t-1)\frac{1-\delta(t)\alpha(t-1)^2}{1-\alpha(t)^2} = \alpha(t-1).$$

Therefore

$$\begin{aligned}\mu_t^{\mathrm{DDPM}} &= A_t\hat{x}_{0,t} + B_t x_t \\ &= \left(A_t + B_t\alpha(t)\right)\hat{x}_{0,t} + B_t\left(x_t - \alpha(t)\hat{x}_{0,t}\right) \\ &= \alpha(t-1)\hat{x}_{0,t} + B_t\left(x_t - \alpha(t)\hat{x}_{0,t}\right) \\ &= c_t^{\mathrm{Tw}} + B_t\left(x_t - \alpha(t)\hat{x}_{0,t}\right).\end{aligned}$$

The Tweedie kernel uses noise scale $\beta(t-1)$, while the DDPM ancestral kernel uses posterior noise scale

$$\tau_t^2 = \frac{\left(1-\delta(t)\right)\left(1-\alpha(t-1)^2\right)}{1-\alpha(t)^2}.$$

This proves both the center decomposition and the variance difference. □

Proposition H.2 shows that DDPM differs from Tweedie in two ways. First, it uses a different Gaussian noise scale. Second, its center contains the residual $r_t = x_t - \alpha(t)\hat{x}_{0,t}$. If x_t carries corrupted cellwise preferences, this term is harmful and may even destabilize the trajectory. The next proposition formalizes this failure mode.

Proposition H.3 (State-memory corruption)**.** *Let*

$$r_t = x_t - \alpha(t)\hat{x}_{0,t}.$$

Assume the denoiser prediction decodes to a valid grid $v \in \mathcal{V}$:

$$D(\hat{x}_{0,t}) = v.$$

For a cell i and competitor $a \neq v_i$, define the denoiser margin

$$M_{i,a}^{\theta} = \hat{x}_{0,t,i,v_i} - \hat{x}_{0,t,i,a},$$

and the residual anti-margin

$$M_{i,a}^{r} = r_{t,i,a} - r_{t,i,v_i}.$$

If for some i, a,

$$B_t M_{i,a}^{r} - \alpha(t-1)M_{i,a}^{\theta} \geq \gamma_{\mathrm{bad}} > 0,$$

then the DDPM center prefers symbol a over the proposal's symbol v_i in cell i with margin at least γ_{bad}:

$$c_{t,i,a}^{\mathrm{DDPM}} - c_{t,i,v_i}^{\mathrm{DDPM}} \geq \gamma_{\mathrm{bad}}.$$

If every valid completion has cell i equal to v_i, then

$$D(c_t^{\mathrm{DDPM}}) \notin \mathcal{V}.$$

Under this assumption,

$$\mathbb{P}\left(D(x_{t-1}^{\mathrm{DDPM}}) \in \mathcal{V} \mid x_t\right) \leq \Phi\left(-\frac{\gamma_{\mathrm{bad}}}{\sqrt{2}\,\tau_t}\right).$$

Proof. The DDPM center is

$$c_t^{\mathrm{DDPM}} = \alpha(t-1)\hat{x}_{0,t} + B_t r_t.$$

Therefore

$$\begin{aligned}c_{t,i,a}^{\mathrm{DDPM}} - c_{t,i,v_i}^{\mathrm{DDPM}} &= \alpha(t-1)\left(\hat{x}_{0,t,i,a} - \hat{x}_{0,t,i,v_i}\right) + B_t\left(r_{t,i,a} - r_{t,i,v_i}\right) \\ &= -\alpha(t-1)M_{i,a}^{\theta} + B_t M_{i,a}^{r}.\end{aligned}$$

By assumption, this is at least γ_{bad}.

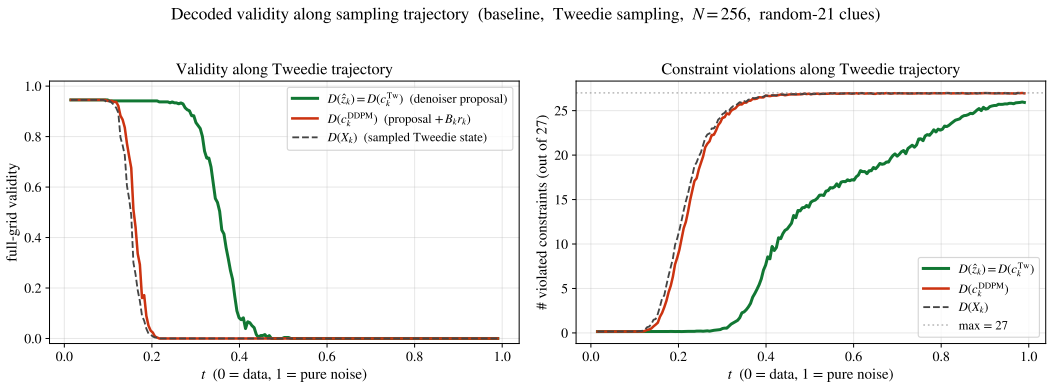


Figure 4: **Decoded validity along a Tweedie sampling trajectory.** For each reverse step, we decode the denoiser proposal $\hat{x}_{0,t}$, the Tweedie center $c_t^{\mathrm{Tw}} = \alpha(t-1)\hat{x}_{0,t}$, and the counterfactual DDPM center $c_t^{\mathrm{DDPM}} = c_t^{\mathrm{Tw}} + B_t r_t$. Left: full-grid Sudoku validity. Right: number of violated constraints. The Tweedie center shares the proposal's argmax, while the DDPM residual anchor delays validity.

If all valid completions have symbol v_i in cell i, then any decoded grid with a different symbol in cell i is invalid. For the noisy DDPM step to become valid, v_i must overtake a. This requires

$$\tau_t \epsilon_{i,v_i} - \tau_t \epsilon_{i,a} \geq \gamma_{\mathrm{bad}}.$$

The left-hand side is Gaussian with variance $2\tau_t^2$, giving

$$\mathbb{P}(\text{repair}) \leq \Phi\left(-\frac{\gamma_{\mathrm{bad}}}{\sqrt{2}\,\tau_t}\right).$$

□

The proposition suggests that DDPM kernel can be centered on the wrong decoded symbol whenever the state residual exceeds the denoiser's correction:

$$B_t\,(r_{t,i,a} - r_{t,i,v_i}) > \alpha(t-1)\,(\hat{x}_{0,t,i,v_i} - \hat{x}_{0,t,i,a})\,.$$

In that case, sampling more precisely around the DDPM center does not help: the sampler is concentrated around the wrong decoded state. We next visualize this effect along actual reverse trajectories.

Figures 4-5 show what happens during sampling. Along the Tweedie trajectory, the model begins to predict valid Sudoku grids before the end of the reverse process. The important point is that Tweedie does not anchor the next center to the current decoded grid. Instead, it recenters directly at $c_t^{\mathrm{Tw}} = \alpha(t-1)\hat{x}_{0,t}$. Since multiplication by the positive scalar $\alpha(t-1)$ does not change the argmax, the Tweedie center has the same decoded grid as the model prediction. Therefore, if the current sample is near-valid but has a few wrong cells, and the denoiser predicts a valid completion, Tweedie can move directly to that valid completion.

DDPM behaves differently. Its center also contains the residual term $B_t r_t$, which keeps part of the current state. This can be helpful when the current state is already correct, but it can be harmful when the current state is near-valid with a few persistent mistakes. In that case, the DDPM update averages the denoiser correction with the current wrong cell preferences, so the sampler can reduce the number of constraint violations while still failing to make the final argmax changes needed for full validity.

Figure 6 shows that adding more exploration does not fix the baseline DDPM sampler. Both samplers degrade when the noise is too small or too large, but the best DDPM setting still remains far below the best Tweedie setting.

To check whether the residual term helps or hurts in practice, we compare the decoded Tweedie and DDPM centers at each reverse step. Let E_+ be the event that the Tweedie center is valid but the DDPM center is invalid, and let E_- be the opposite event. If the residual anchor were often useful for Sudoku validity, then E_- should occur frequently. Instead, Figure 2a shows that E_+ dominates.

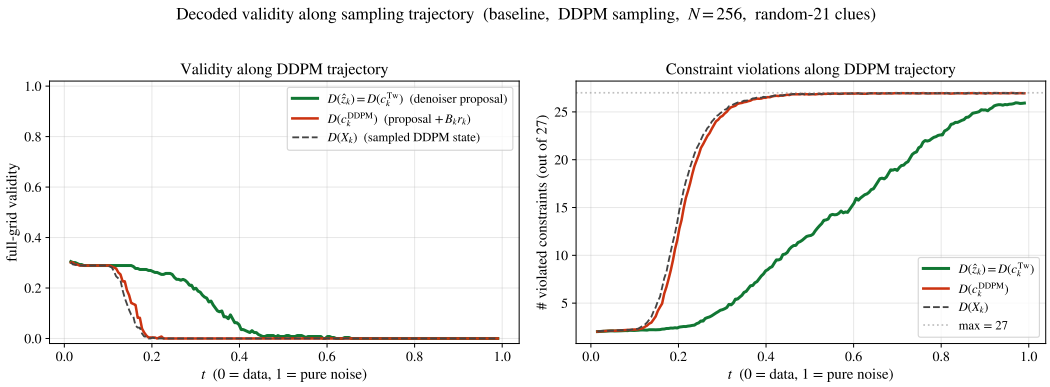


Figure 5: **Decoded validity along a DDPM sampling trajectory.** Same diagnostics as Fig. 4, but the trajectory itself is generated by DDPM ancestral sampling. The number of violated constraints decreases along the trajectory, showing that the sampler moves toward Sudoku structure. However, full-grid validity remains low because a small number of persistent errors is enough to invalidate the decoded grid. This illustrates the state-anchor failure mode: DDPM can approach a near-valid solution while still failing to make the needed cellwise corrections required for exact validity.

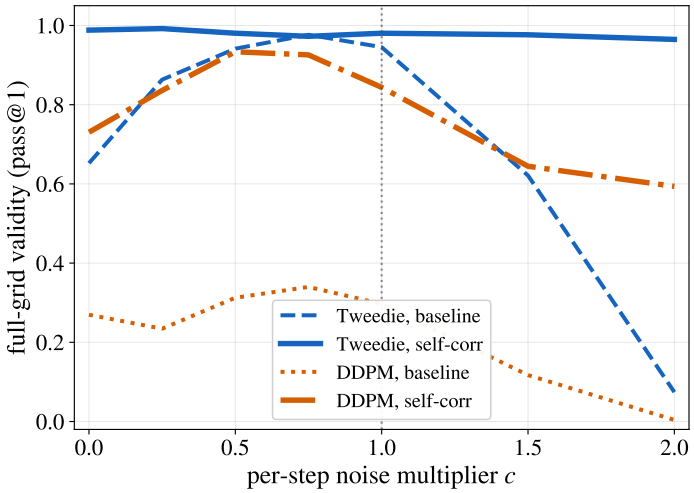


Figure 6: **Effect of DDPM sampling variance on Sudoku.** Final validity when multiplying the per-step sampling variance. Changing the variance alone does not close the baseline DDPM–Tweedie gap.

H.4 Empirical comparison of training inputs and sampler states

We next use the same E_+ diagnostic to compare forward-noised training inputs X_t^{std}, self-correction inputs X_t^{SC}, and actual DDPM states Y_t. The two training inputs are constructed as

$$X_t^{\mathrm{std}} = \alpha(t)x_0 + \beta(t)\epsilon, \qquad X_t^{\mathrm{SC}} = \alpha(t)\hat{x}_0 + \beta(t)\epsilon',$$

where x_0 is a valid training sample, $\hat{x}_0$ is the model's intermediate prediction constructed as in Section 3.3, and ϵ, ϵ' are independent standard Gaussian noise draws. The states Y_t are obtained along DDPM sampling trajectories.

The respective event frequencies are

$$(0.319,\ 0.116,\ 0.030) \quad \text{on Sudoku-Extreme,}$$

and

$$(0.247,\ 0.067,\ 0.036) \quad \text{on 21-clue Sudoku.}$$

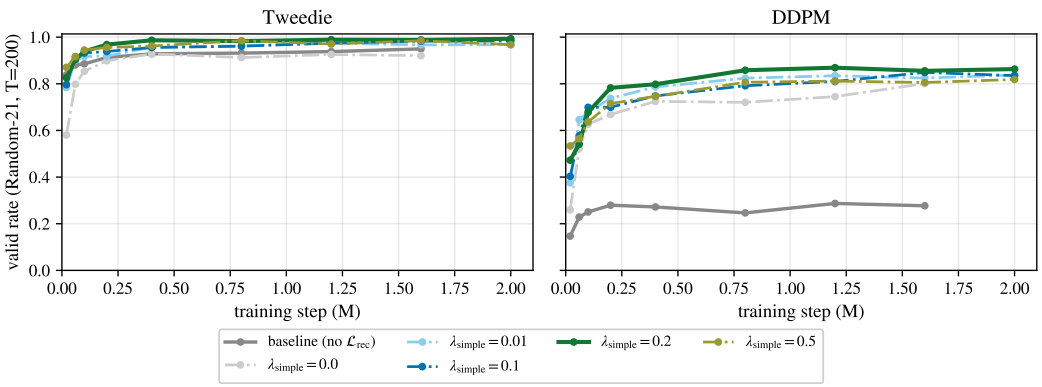


Figure 7: **Self-correction loss-weight sensitivity on Sudoku puzzles with 21 clues.** We sweep $\lambda_{\text{simple}} \in \{0, 0.01, 0.1, 0.2, 0.5\}$ and report checkpoint trajectories under the standard $T = 200$, pass@1 evaluation.

Thus, on this diagnostic, the gap between self-correction inputs and DDPM states is 3.4–$6.8\times$ smaller than for standard forward-noised inputs. This supports the interpretation that self-correction inputs better reflect sampler states on this center-disagreement statistic, without establishing closeness of the full input distributions.

I Self-Correction Loss Ablations on Sudoku

We ablate two design dimensions of the self-correction loss $\mathcal{L}_{\text{SC}}$ on Sudoku: (i) the regularizer loss weight λ_{simple}; and (ii) alternative formulations of the self-correction training step, including different input constructions, target constructions, and multi-step rollouts. We additionally compare to input perturbations from DDPM-IP (Ning et al., 2023) and self-conditioning from Analog Bits (Chen et al., 2022).

We evaluate on Sudoku puzzles with 21 clues using the full checkpoint grid $\{20\text{k}, 60\text{k}, 100\text{k}, 200\text{k}, 400\text{k}, 800\text{k}, 1.2\text{M}, 1.6\text{M}, 2.0\text{M}\}$. For certain runs we stopped the training earlier if loss was unstable. For the loss-weight sweep, we report both samplers: Tweedie reprojection (Tw) and DDPM.

I.1 Simple loss weight

Figure 7 shows the sweep over $\lambda \in \{0, 0.01, 0.1, 0.2, 0.5\}$. The baseline corresponds to standard training without the self-correction loss. Tweedie reprojection already performs strongly across settings, with valid rates in a relatively narrow range. The main effect of the self-correction loss is on DDPM. As discussed in Section H.3, DDPM preserves information from the current state x_t directly. This is appropriate for continuous denoising, but in discrete constraint problems it can preserve an early incorrect commitment. Self-correction training mitigates this failure mode, improving the validity of DDPM by approximately 55 percentage points on Sudoku puzzles with 21 clues.

Overall, the method is not sensitive to the exact value of λ. In the main experiments we use $\lambda_{\text{simple}} = 0.1$, which was chosen as the initial default and lies in the stable high-performing region of the sweep.

I.2 Loss-Formulation Variants

We next compare alternative ways of constructing the self-correction training step.

Baseline. No self-correction loss. This is standard single-pass denoising training (as in I.1).

Self-correction. Our headline recipe described in Algorithm 2. The model first predicts $\hat{x}_0$, then receives a corrupted version of this previous prediction and is trained to recover the original clean target x_0.

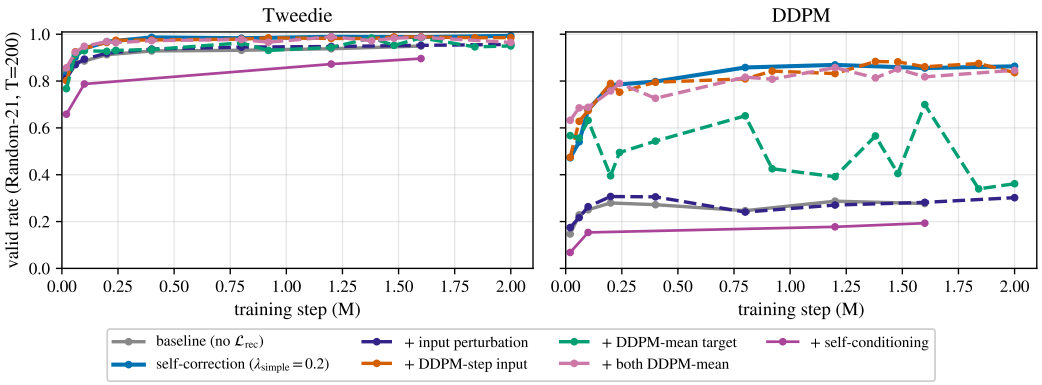


Figure 8: **Loss ablations on Sudoku puzzles with 21 clues.** Self-correction and DDPM-step input give the strongest and most stable improvements, while input perturbation and self-conditioning are substantially weaker, especially for DDPM.

Input perturbation. A DDPM-IP-style input regularization baseline from Ning et al. (2023). The input perturbation baseline adds extra Gaussian noise to the input:

$$\tilde{x}_t = x_t + \gamma\epsilon'', \qquad \gamma = 0.1, \qquad \epsilon'' \sim \mathcal{N}(0, I),$$

and trains on

$$\|f_\theta(\tilde{x}_t, t) - x_0\|^2 .$$

This encourages robustness to local perturbations of x_t, but it does not specifically train the model to correct structured errors arising from its own previous predictions.

DDPM-step input. Instead of constructing the second input by forward-noising $\hat{x}_0$, we construct it using one DDPM ancestral step from x_{t_1}. Concretely, after computing $\hat{x}_0 = f_\theta(x_{t_1}, t_1)$, we set

$$x_{t_2} = \mu_{\text{DDPM}}(x_{t_1}, \hat{x}_0; t_1 \to t_2) + \tau_{t_1,t_2}\xi, \qquad \xi \sim \mathcal{N}(0, I),$$

where μ_{DDPM} is the ancestral DDPM mean and τ_{t_1,t_2} is the corresponding posterior noise scale. This makes the self-correction input distribution closer to the test-time DDPM trajectory. The loss itself is unchanged from the standard self-correction loss $\hat{x}_0'' = f_\theta(x_{t_2}, t_2)$, $\|\hat{x}_0'' - x_0\|^2$ only the input distribution is modified.

DDPM-mean target. Instead of supervising the second prediction directly with $\|\hat{x}_0' - x_0\|^2$, we supervise the resulting DDPM mean:

$$\|\mu_{\text{DDPM}}(x_{t_2}, \hat{x}_0'; t_2 \to t_3) - \alpha(t_3)x_0\|^2 .$$

This asks the model to make predictions whose downstream DDPM update lands on the clean manifold.

DDPM-step input + DDPM-mean target. Combines the previous two modifications: the self-correction input is generated by a DDPM step, and the loss supervises the downstream DDPM mean.

Random-N denoise. Performs a random number of no-gradient self-correction rollouts before the final gradient pass. With parameter n, the number of rollouts is sampled from $\{1, \ldots, n\}$. This exposes the model to deeper unrolled trajectories.

Self-conditioning without self-correction loss. Following Chen et al. (2022), the model receives its previous prediction as an additional input channel, $f_\theta(\text{cat}([x_t, \hat{x}_0])_{\text{dim}=-1}, t)$, no self-correction loss.

Figure 8 compares alternative ways of constructing the recovery signal on Sudoku puzzles with 21 clues. Input perturbation gives little improvement over the baseline, suggesting that generic Gaussian noise does not reproduce the structured errors created by closed-loop sampling. Self-conditioning is also weaker, especially for DDPM, indicating that simply providing an additional memory channel is not enough. The strongest variants are those that expose the model to its own intermediate predictions,

Table 7: **Ablation on invalid-state exposure for Sudoku conditional generation.** We evaluate 21-given-cell Sudoku with 500 sampling steps on 1000 puzzles and 3 seeds (checkpoint at 0.2M steps). Values are valid rates in percent, mean ± std.

Training variant	Tweedie reprojection	DDPM
Baseline	99.23 ± 0.35	34.47 ± 1.75
Random-symbol corruption, 20% cells	99.60 ± 0.10	64.00 ± 2.33
Random-symbol corruption, 40% cells	99.07 ± 0.31	56.60 ± 1.23
Self-correction	99.07 ± 0.31	78.20 ± 0.98

Table 8: **Sudoku validity with rectified flow.** Entries report baseline → self-correction for the additional rectified-flow experiment.

Regime	Euler	EM decay	Tweedie
21 clues	$0.155 \to 0.630$	$0.427 \to 0.926$	$0.926 \to 0.988$
Medium	$0.759 \to 0.909$	$0.864 \to 0.960$	$0.983 \to 0.993$
Hard	$0.121 \to 0.735$	$0.417 \to 0.963$	$0.911 \to 0.995$

either through the simple self-correction loss or DDPM-aware variants. This supports our main interpretation: the issue is not only robustness to noise or lack of conditioning, but a mismatch between the forward-noised training inputs and the model-induced states visited during sampling.

Random-symbol corruption. We also test whether self-correction helps only by exposing the model to invalid discrete inputs. As a simpler baseline, we randomly select either 20% or 40% of Sudoku cells, replacing each selected cell with a random symbol. We then train the model to recover the original valid grid. This improves DDPM from 34.5% to 64.0% with 20% corrupted cells, showing that invalid-state exposure is useful. However, self-correction improves DDPM further to 78.2%, suggesting that model-induced recovery states are better matched to the reverse-sampling distribution than arbitrary symbol corruptions. Tweedie reprojection is already near saturation in this setting, so these augmentations mainly affect DDPM.

Training on multi-step DDPM trajectories. We additionally tested whether training on states from an actual trajectory improves over the one-step self-correction loss. Starting from a converged denoiser, we maintained a pool of 16-step DDPM trajectories and advanced each trajectory by two differentiable reverse steps per update using truncated backpropagation through time. On 21-clue Sudoku, the best rollout-trained model reaches 0.78 DDPM validity, compared with 0.31 for standard training and 0.87 for one-step self-correction. Thus, training on trajectory states helped, but did not outperform one-step self-correction.

I.3 Rectified-flow experiment on one-hot Sudoku

In addition to evaluating the released SRM checkpoint, we train a rectified-flow model on one-hot Sudoku with random conditioning masks. Table 8 compares sampling with and without self-correction training. The same qualitative pattern appears: Tweedie reprojection improves over Euler without retraining, while self-correction substantially improves Euler.

I.4 Consistency-model experiment

We additionally evaluate a consistency model on conditional one-hot Sudoku-Extreme. We train the model from scratch using an adaptation of improved consistency training (Song & Dhariwal, 2024). It uses the same VP-cosine path and 0.82M-parameter Transformer architecture as our continuous baseline, with hidden dimension 128 and depth 4. Training uses the dataset's given-clue masks, a progressively increasing number of noise discretization levels, a stopped-gradient target without exponential moving averaging, and a noise-weighted pseudo-Huber loss. We train for 2 million optimizer steps with a learning rate of 10^{-4} and warmup. This consistency model achieves approximately 3% pass@1 validity with 5 sampling steps and 16% with $1{,}000$ steps. For context,

Table 9: **Sampling step-count ablation.** Full-grid Sudoku validity for baseline and self-correction models under Tweedie and DDPM sampling as the number of sampling steps T varies. Values are mean $\pm$ standard error over runs (different seeds for random masks generation).

	Tweedie		DDPM	
T	Baseline	Self-correction	Baseline	Self-correction
25	0.488 ± 0.014	0.355 ± 0.022	0.212 ± 0.036	0.441 ± 0.034
50	0.737 ± 0.018	0.887 ± 0.027	0.272 ± 0.013	0.745 ± 0.030
100	0.853 ± 0.011	0.965 ± 0.007	0.249 ± 0.020	0.848 ± 0.014
200	0.952 ± 0.019	0.978 ± 0.006	0.296 ± 0.033	0.852 ± 0.017
500	0.987 ± 0.005	0.992 ± 0.007	0.292 ± 0.062	0.861 ± 0.025
1000	0.999 ± 0.002	0.999 ± 0.002	0.311 ± 0.020	0.854 ± 0.024
2000	1.000 ± 0.000	1.000 ± 0.000	0.298 ± 0.022	0.852 ± 0.000
5000	1.000 ± 0.000	1.000 ± 0.000	0.292 ± 0.033	0.863 ± 0.036

Table 10: Comparison across Sudoku representations. Results report baseline $\rightarrow$ self-corrected validity (one-seed results).

Experiment	Parameters	DDPM	EM-decay	Tweedie
One-hot Sudoku	824,073	$0.31 \rightarrow 0.87$	$0.84 \rightarrow 0.98$	$0.94 \rightarrow 0.98$
Analog-bit Sudoku	822,788	$0.19 \rightarrow 0.79$	$0.34 \rightarrow 0.87$	$0.89 \rightarrow 0.98$
Random-embedding Sudoku	824,073	$0.26 \rightarrow 0.76$	$0.43 \rightarrow 0.90$	$0.89 \rightarrow 0.93$
mini MNIST–Sudoku	838,208	$0.01 \rightarrow 0.39$	$0.01 \rightarrow 0.44$	$0.42 \rightarrow 0.78$

at $1{,}000$ steps, our baseline diffusion model achieves 18% with EM decay and 26% with Tweedie reprojection.

J Sampling step-count ablation

As an additional sanity check, we vary the number of reverse sampling steps T for Sudoku puzzles with 21 clues. This tests whether the DDPM gap is simply due to insufficient discretization or too few opportunities to correct errors.

As Table 9 shows, increasing T drives Tweedie sampling to near-perfect validity, but baseline DDPM remains around 0.29–0.31. Thus the baseline DDPM failure is not fixed by using more reverse steps.

K Beyond one-hot encodings

To verify that our findings are not specific to one-hot representations and argmax decoding, we vary both the representation and the decoder while using models of comparable size. We evaluate uint4 analog codes with threshold decoding (adapted from uint8 Analog Bits (Chen et al., 2022)), fixed random embeddings with nearest-neighbour decoding, and mini MNIST-Sudoku with a learned CNN decoder (different from the dataset used by SRM (Wewer et al., 2025)). For mini MNIST-Sudoku, we use 8×8 MNIST images, whereas SRM uses 28×28 images; our model has 0.8M parameters compared with approximately 130M for SRM. Each image is high-dimensional and continuous, and each cell is decoded by a learned classifier, following SRM. Table 10 contains relevant within-representation comparisons.

We observe consistent improvements from DDPM to Tweedie for both the standard Sudoku checkpoint and the self-correction-trained model, while DDPM itself also benefits from self-correction training.

Table 11: **Distance to one-hot representations across samplers and tasks.** Per-puzzle RMSE between the sampler output x and its nearest one-hot codeword $\mathrm{onehot}(\arg\max x)$, over predicted (non-clue) cells, $\times 10^2$ (lower means closer to one-hot). Mean $\pm$ std over training seeds. Baselines use no self-correction loss; self-correction uses $\lambda_{\mathrm{simple}} = 0.1$. EM uses a constant noise scale along the trajectory, so its final prediction contains noise in our implementation.

Sampler	Sudoku			Sudoku-Extreme	N-Queens		Latin		GC	
	21 clues	Medium	Hard	pass@1	random	gen	random	gen	$N = 12$	$N = 18$
Baseline (no self-correction loss)										
DDPM	1.5±0.2	0.5±0.1	1.6±0.1	2.3±1.0	0.5±0.0	0.5±0.0	1.2±0.1	1.0±0.1	1.1±0.1	1.1±0.1
Euler	2.1±0.3	0.6±0.1	2.4±0.1	3.3±0.5	0.5±0.0	0.5±0.0	1.8±0.1	1.6±0.0	1.4±0.1	1.3±0.1
EM	9.7±0.0	8.9±0.0	10.2±0.1	6.3±0.1	8.7±0.0	8.7±0.0	9.9±0.1	10.2±0.1	9.7±0.2	9.5±0.1
EM decay	0.4±0.0	**0.3±0.0**	0.7±0.1	1.6±0.5	0.5±0.0	**0.5±0.1**	0.4±0.0	0.4±0.0	0.6±0.3	**0.6±0.3**
Tweedie reprojection	**0.3±0.0**	**0.3±0.0**	**0.4±0.0**	**1.2±0.4**	**0.4±0.1**	**0.5±0.0**	**0.3±0.0**	**0.3±0.0**	**0.5±0.2**	**0.6±0.2**
Self-correction loss (with $\lambda_{\mathrm{simple}} = 0.1$)										
DDPM	0.9±0.1	0.4±0.1	1.6±0.3	3.0±0.3	1.2±0.1	4.3±0.1	0.7±0.1	1.6±0.3	0.5±0.4	0.6±0.2
Euler	1.8±0.1	0.7±0.1	3.2±0.1	4.1±0.5	1.8±0.2	6.1±0.3	1.4±0.2	5.4±0.3	0.5±0.2	0.6±0.2
EM	9.7±0.1	9.0±0.0	11.4±0.3	5.1±0.4	9.5±0.1	13.8±0.4	9.3±0.1	13.2±1.4	1.1±0.1	9.0±0.1
EM decay	**0.3±0.1**	**0.3±0.0**	**0.4±0.1**	1.9±0.2	0.7±0.1	**2.0±0.4**	**0.4±0.1**	**0.5±0.1**	**0.5±0.3**	**0.4±0.2**
Tweedie reprojection	**0.3±0.1**	0.4±0.1	0.7±0.2	**1.1±0.2**	**0.6±0.0**	2.9±0.2	**0.4±0.1**	0.8±0.3	0.7±0.3	0.7±0.3

L Other metrics

L.1 Soft metrics

The main results use exact validity, which is a strict binary metric: a solution with a single violated constraint is counted as invalid. We therefore report two additional diagnostics.

Distance to one-hot (Table 11) is the per-puzzle RMSE between the final continuous output and its nearest one-hot encoding, computed over predicted cells. Lower values indicate that the final continuous state is closer to an exact discrete representation, but do not imply that the decoded configuration is valid. We report this metric for all settings except Sudoku-Extreme pass@10, which reuses the pass@1 boards.

We also report the *constraint-violation rate* (Table 12): the fraction of predicted cells involved in a violated task constraint, using row/column/box constraints for Sudoku, row/column constraints for Latin squares, and attacking-queen constraints for N-queens. Overall, constraint-violation rate shows largely the same trends as exact validity.

L.2 Uniqueness and coverage

High validity alone does not rule out repeatedly generating the same solutions. We therefore measure *valid uniqueness*: the number of distinct valid decoded boards divided by the number of valid generated boards. We compute this ratio separately for each training seed and then average across seeds. Table 13 reports the results. Duplicates are uncommon in these evaluations. For conditional tasks, however, different samples have different masks and clues, so uniqueness across the batch does not establish diversity for a fixed conditioning input.

For $N = 14$, the complete solution space contains $365{,}596$ boards. The training set contains $168{,}000$ distinct boards, leaving $197{,}596$ solutions outside the training set. For each configuration, we generate $500{,}000$ unconditional samples and count the distinct valid boards. We report total coverage as well as coverage within and outside the training set, using the size of each set as the corresponding denominator.

Self-correction increases total coverage from 20.0% to 44.8% with EM decay and from 24.6% to 33.8% with Tweedie. Coverage is similar within and outside the training set. Thus, the validity gains are accompanied by broader coverage of both training and unseen solutions, rather than repeated generation of a small set of training boards.

Table 12: **Constraint violations across samplers and tasks.** Percentage of denoised cells that violate a task constraint (lower is better): row, column, or box constraints for Sudoku; row or column constraints for Latin squares; and attacking-queen constraints for N-Queens. Entries are mean ± standard deviation across training seeds. Baselines use no self-correction loss; self-correction uses $\lambda_{\text{simple}} = 0.1$.

Sampler	Sudoku			Sudoku-Extreme	N-Queens		Latin	
	21 clues	Medium	Hard	pass@1	random	gen	random	gen
Baseline (no self-correction loss)								
DDPM	4.5±0.1	1.3±0.2	3.7±0.1	8.7±0.4	13.1±0.0	24.6±0.7	2.8±0.6	1.1±0.4
Euler	6.0±0.2	1.8±0.1	5.3±0.0	9.6±0.9	17.6±0.8	29.1±2.8	3.9±0.1	2.1±0.3
EM	5.8±0.0	1.8±0.0	4.6±0.0	10.0±0.3	17.3±1.0	27.4±2.4	3.4±0.3	1.9±0.5
EM decay	0.9±0.0	0.3±0.0	1.0±0.1	4.7±0.1	7.1±0.3	18.9±3.0	0.3±0.1	0.1±0.1
Tweedie reprojection	**0.3±0.1**	**0.1±0.0**	**0.4±0.0**	**4.0±0.1**	**4.7±1.0**	**13.6±0.3**	**0.1±0.1**	**0.0±0.0**
Self-correction loss (with $\lambda_{\text{simple}} = 0.1$)								
DDPM	0.8±0.1	0.3±0.0	1.3±0.3	6.5±0.3	6.5±0.1	12.5±0.2	0.2±0.1	0.9±0.4
Euler	1.6±0.2	0.5±0.0	3.8±0.3	8.1±0.7	9.6±1.0	17.5±1.9	0.7±0.3	5.2±0.6
EM	1.4±0.2	0.5±0.0	2.5±0.5	7.7±0.3	8.7±1.1	17.1±0.1	0.4±0.2	3.2±1.2
EM decay	**0.1±0.0**	**0.0±0.0**	**0.1±0.0**	3.5±0.1	3.1±0.2	**7.5±0.8**	**0.0±0.0**	**0.0±0.0**
Tweedie reprojection	**0.1±0.1**	**0.0±0.0**	0.4±0.1	**2.7±0.2**	**2.5±0.2**	9.3±0.5	**0.0±0.0**	0.3±0.3

Table 13: **Uniqueness among valid generated boards.** Entries are percentages, shown as baseline → self-correction and averaged across training seeds. Standard deviations are omitted. Random denotes random conditioning; generation denotes unconditional sampling.

Sampler	Sudoku 21 clues	Sudoku Medium	Sudoku Hard	Sudoku-Extreme
DDPM	100.0 → 100.0	100.0 → 99.9	100.0 → 100.0	100.0 → 99.8
Euler	100.0 → 100.0	99.9 → 99.9	100.0 → 100.0	100.0 → 100.0
EM	100.0 → 100.0	100.0 → 99.9	100.0 → 100.0	100.0 → 99.9
EM decay	100.0 → 100.0	99.8 → 99.9	100.0 → 100.0	99.7 → 99.7
Tweedie reprojection	100.0 → 100.0	99.9 → 99.9	100.0 → 100.0	99.9 → 99.7
Sampler	N-Queens random	N-Queens generation	Latin random	Latin generation
DDPM	97.0 → 96.5	100.0 → 99.8	100.0 → 100.0	100.0 → 100.0
Euler	98.2 → 97.0	100.0 → 100.0	100.0 → 100.0	100.0 → 100.0
EM	100.0 → 100.0	100.0 → 100.0	100.0 → 100.0	100.0 → 100.0
EM decay	100.0 → 100.0	100.0 → 100.0	100.0 → 100.0	100.0 → 100.0
Tweedie reprojection	96.0 → 95.7	99.8 → 99.5	100.0 → 100.0	100.0 → 100.0

L.3 Multiple completions for the same Sudoku clues

We also test whether the model can generate different valid completions for a fixed conditioning input. We select 50 Sudoku puzzles with 32–38 revealed cells and enumerate all valid completions by backtracking. Each puzzle admits between 2 and 12 solutions, with 297 solutions in total. For each puzzle, we run Tweedie reprojection with 1,000 independent noise realizations under baseline and self-correction training. Both configurations recover 139 of the 297 solutions, or 46.8%. Thus, training against individual clean targets does not restrict sampling to a single completion for each set of clues. However, neither configuration recovers all possible completions, and self-correction does not improve the aggregate coverage in this experiment.

Table 14: **Coverage of the complete $N = 14$ Queens solution space.** Each configuration uses 500,000 unconditional samples. The two count columns report distinct valid generated boards. Coverage percentages use denominators 365,596, 168,000, and 197,596 for the full, training, and outside-training sets, respectively.

Sampler	Training	Total coverage (%)	Distinct in training	Distinct outside training	Training coverage (%)	Outside coverage (%)
EM decay	Baseline	20.0	33,523	39,629	20.0	20.1
EM decay	Self-correction	44.8	75,360	88,436	44.9	44.8
Tweedie reprojection	Baseline	24.6	41,274	48,572	24.6	24.6
Tweedie reprojection	Self-correction	33.8	57,362	66,063	34.1	33.4

M TRAJECTORIES UNDER DIFFERENT SAMPLERS

When does Tweedie reprojection fail? We distinguish failures according to whether a valid clean proposal appears during sampling. A failure is denoiser-related when the final Tweedie output is invalid and the decoded clean proposal was never valid at any recorded reverse step. In a diagnostic batch of 256 unconditional N-Queens trajectories, all failures are of this type. On Sudoku-Extreme, 38% of Tweedie failures are denoiser-related. In the remaining 62%, a valid proposal appears at least once, but the trajectory still ends in an invalid sample. Poor proposal quality is therefore one limitation of Tweedie reprojection, but finding a valid proposal during sampling does not by itself guarantee a valid final output.

Visualizations. We show example sampling trajectories for Latin squares (Figure 9), N-queens (Figure 10), graph connectivity (Figure 11), and Sudoku (Figure 12). Within each task, all sampler rows use the same trained checkpoint; different tasks use different models. For MNIST-Sudoku, we also show trajectories from the released SRM checkpoint (Wewer et al., 2025) under Euler sampling (equivalently, deterministic DDIM on the linear path; bottom) and Tweedie reprojection (top), visualizing both x_t (Figure 13) and the corresponding clean predictions $\hat{x}_0$ (Figure 14). In both cases, individual cells resemble recognizable MNIST digits, which we refer to as local correctness. However, the DDIM trajectory does not produce a globally valid Sudoku grid; for example, the central 3×3 block contains the digit 1 twice. In the same example, Tweedie reprojection ends in a valid solution.



Figure 9: Latin squares trajectory

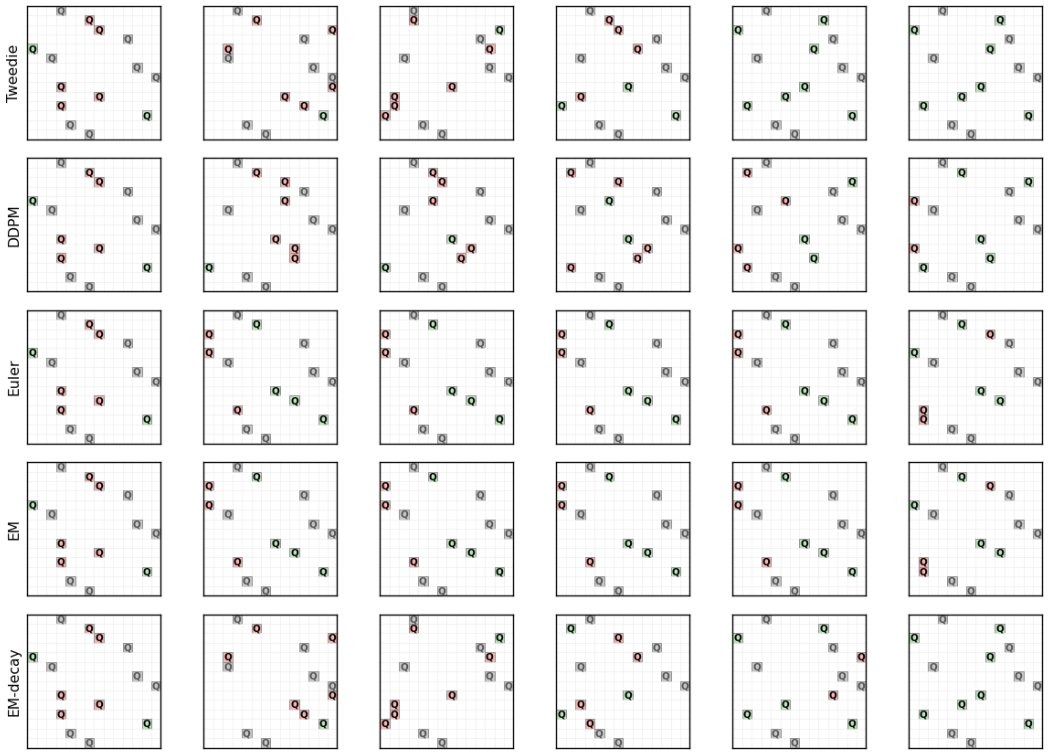


Figure 10: N-Queens trajectory

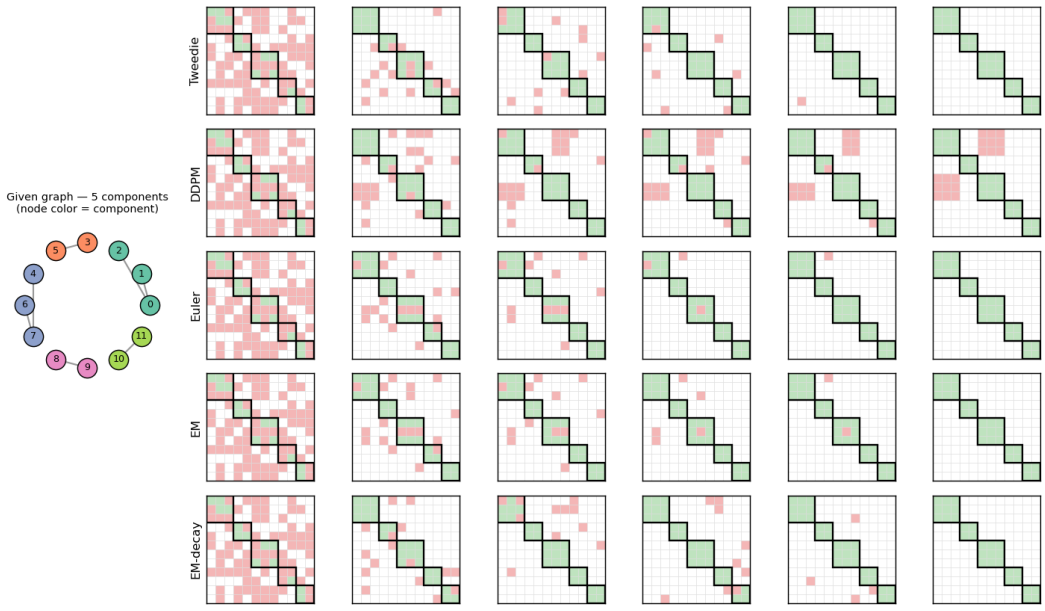


Figure 11: Graph connectivity trajectory

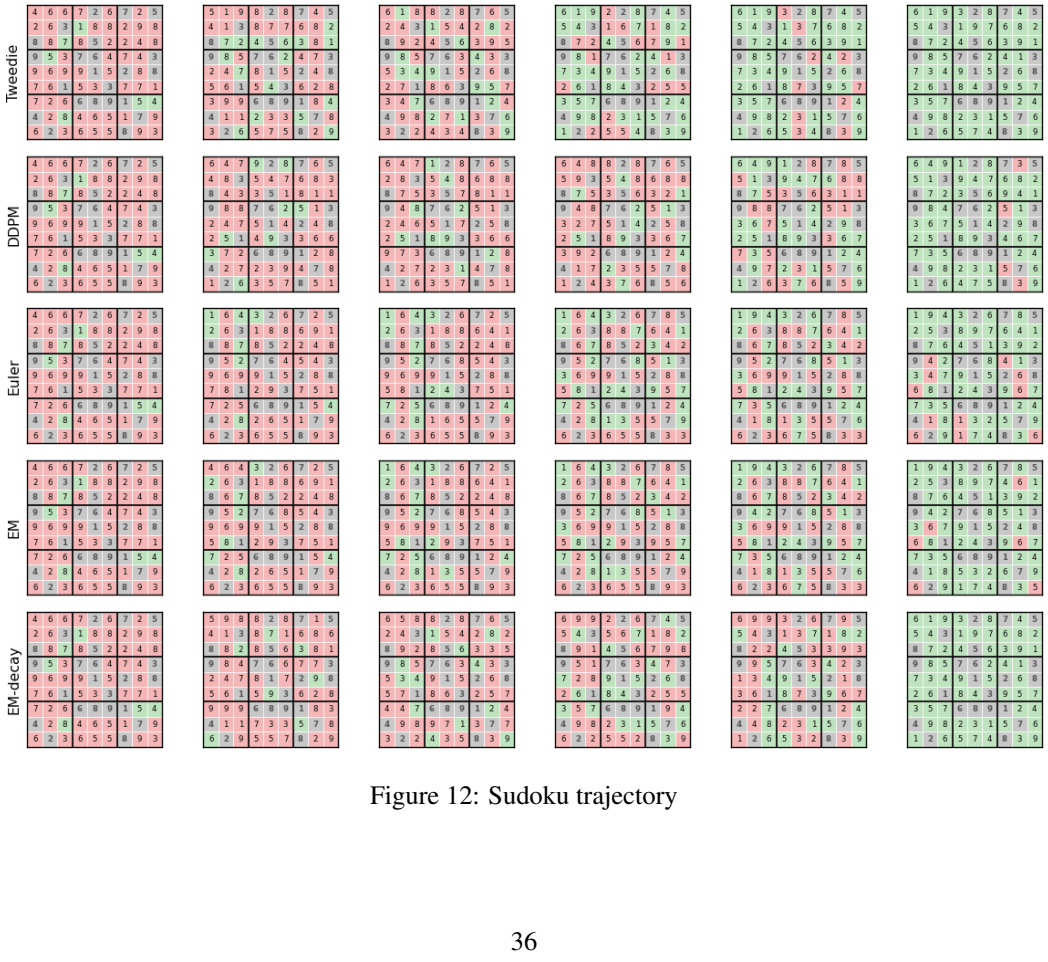


Figure 12: Sudoku trajectory

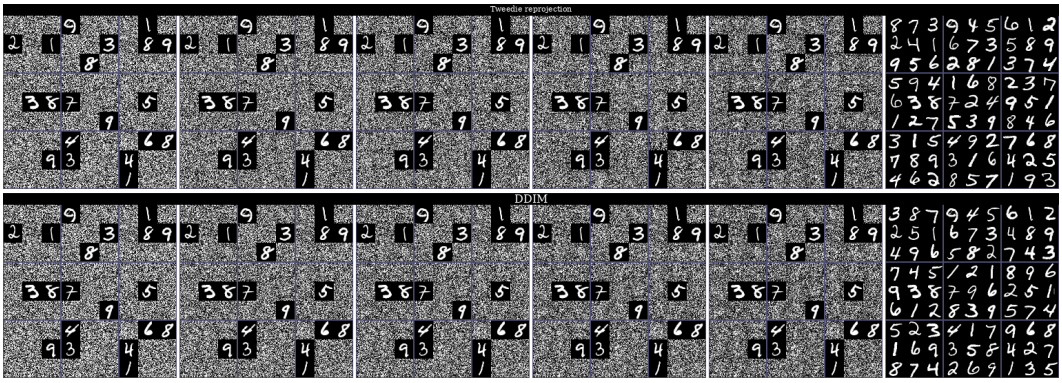


Figure 13: Sudoku-MNIST trajectory (x_t)

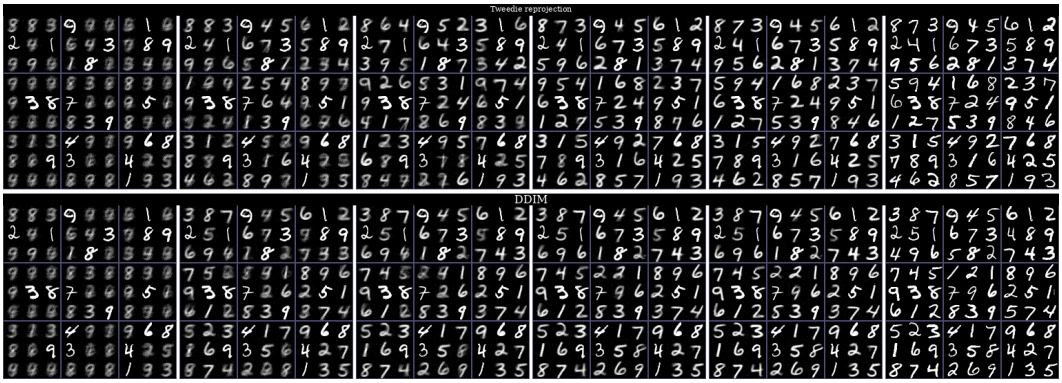


Figure 14: Sudoku-MNIST trajectory ($\hat{x}_0$)