Normalizing Trajectory Models

Jiatao Gu[1]* **Tianrong Chen**[1] **Ying Shen**[2,*] **David Berthelot**[1] **Shuangfei Zhai**[1]
Josh Susskind[1]
[1]Apple [2]UIUC



Figure 1: **NTM generation: four predictor transitions plus one denoiser pass.** We show samples from models trained from scratch at 256×256, and from models obtained by finetuning pretrained flow-matching checkpoints at 512×512.

Abstract

Gaussian reverse transitions become restrictive when diffusion sampling is compressed to a few coarse steps. We introduce Normalizing Trajectory Models (NTM), a framework that models each coarse reverse transition as a conditional normalizing flow. Each transition combines a shallow invertible transporter with a deep spatially parallel predictor, yielding exact transition and joint-trajectory densities. The model can be trained from scratch or initialized from a pretrained flow-matching backbone. Its joint trajectory score supplies targets for a lightweight denoiser that bypasses autoregressive inversion and test-time gradients. Using four predictor transitions and one denoiser pass, the from-scratch model achieves 0.82 GenEval, while the finetuned model achieves 83.38 on DPG-Bench. Exact density evaluation concerns the underlying trajectory model; the learned denoiser provides its fast amortized readout.

*Work done while Jiatao Gu was at Apple with a joint affiliation at the University of Pennsylvania, and Ying Shen was a research intern at Apple.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

1 Introduction

Diffusion-based models [28; 47; 34; 35; 3] have become the dominant paradigm for high-fidelity image generation [43; 17; 41]. Diffusion samplers commonly parameterize reverse steps as Gaussian transitions; flow matching instead learns a velocity field whose ODE is numerically integrated. For sufficiently small diffusion steps, the Gaussian approximation is accurate: the reverse conditional $p(\boldsymbol{x}_s \mid \boldsymbol{x}_t)$ is close to Gaussian because the transition covers only a small portion of the diffusion trajectory. With fewer steps, each stochastic reverse transition spans a larger interval and its true conditional can be multimodal or heavy-tailed. The single-Gaussian assumption then becomes a fundamental bottleneck for few-step generation quality.

A growing body of work addresses the efficiency problem through implicit few-step generators. Distillation methods [45; 55] and consistency models [48; 36] learn to map noise to data in fewer steps, yet provide no tractable density over the generative trajectory. DDGAN [52] replaces the Gaussian reverse with an implicit distribution learned via adversarial training, but introduces mode-seeking behavior and training instability that limit scalability. Our goal is to combine expressive coarse reverse transitions with tractable transition and joint-trajectory densities.

We introduce **Normalizing Trajectory Models (NTM)**, a framework that models $p(\boldsymbol{x}_s \mid \boldsymbol{x}_t)$ as a conditional normalizing flow with exact log-likelihood. The core idea is to learn a latent space—via an invertible transporter—where the reverse conditional becomes simple enough to be modeled by a Gaussian predictor. Unlike a compressive encoder, the transporter preserves dimensionality and invertibility, yielding exact conditional likelihoods through the change-of-variables formula. This connects a predictor–encoder architecture [22; 4] with probabilistic generative modeling.

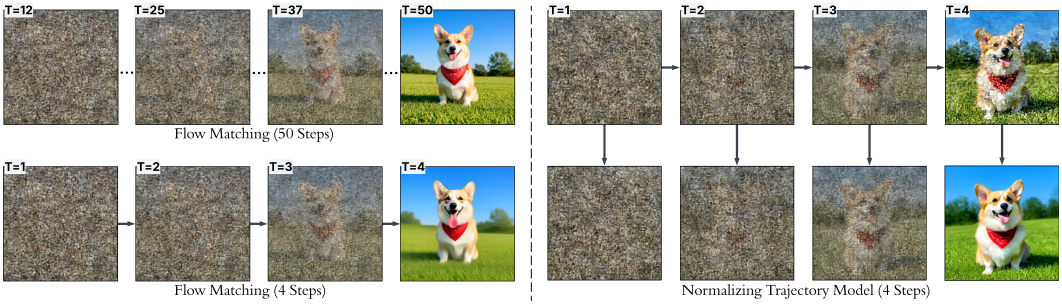


Figure 2: **Denoising trajectories. Left**: Flow matching with 50 steps and 4 steps. **Right**: NTM achieves comparable quality in 4 steps by modeling the non-Gaussian reverse conditional.

NTM can be trained from scratch using stochastic forward trajectories, or initialized from any pretrained flow-matching model by setting the transporter to identity and the predictor to the pretrained Gaussian posterior. The exact trajectory likelihood further enables score-based denoising: since the model is trained on correlated noisy trajectories from a Markov forward process, the gradient of the NTM loss provides a joint score that denoises all timesteps simultaneously by exploiting their correlations. A lightweight learned denoiser amortizes this correction, bypassing spatial autoregressive inversion and test-time gradients. On text-to-image generation, the from-scratch sampler achieves 0.82 GenEval [20] with four predictor transitions and one denoiser pass, compared with 0.56 for STARFlow. Exact density evaluation applies to the underlying transition and trajectory model; the denoiser is its fast amortized readout.

Our contributions are:

- A framework that models the non-Gaussian reverse conditional $p(\boldsymbol{x}_s \mid \boldsymbol{x}_t)$ via an invertible transporter and a Gaussian predictor, yielding exact log-likelihood while bridging representation learning and probabilistic modeling.
- A finetuning recipe that initializes from pretrained diffusion or flow-matching models via identity transporter and zero-initialized scale correction, preserving pretrained quality at initialization.
- Score-based trajectory denoising that exploits the exact likelihood and Markov covariance to jointly correct generated trajectories, amortized by a learned denoiser for fast sampling without additional training data.

2 Preliminaries

2.1 Flow Matching and Diffusion Models

Flow matching [34; 35; 3] defines a forward interpolation between clean data $\boldsymbol{x}_0$ and Gaussian noise $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$:

$$\boldsymbol{x}_t = (1-t)\,\boldsymbol{x}_0 + t\,\boldsymbol{\epsilon}, \quad q(\boldsymbol{x}_t \mid \boldsymbol{x}_0) = \mathcal{N}\big((1-t)\,\boldsymbol{x}_0,\ t^2\boldsymbol{I}\big), \quad t \in [0,1]. \tag{1}$$

A neural network $v_\theta(\boldsymbol{x}_t, t)$ is trained to predict the velocity field by minimizing

$$\mathcal{L}_{\mathrm{FM}} = \mathbb{E}_{t,\,\boldsymbol{x}_0,\,\boldsymbol{\epsilon}}\big\|v_\theta(\boldsymbol{x}_t, t) - (\boldsymbol{\epsilon} - \boldsymbol{x}_0)\big\|^2, \tag{2}$$

and samples are generated by integrating the learned ODE $\mathrm{d}\boldsymbol{x} = v_\theta(\boldsymbol{x}, t)\,\mathrm{d}t$ from $t{=}1$ (noise) to $t{=}0$ (data). Mathematically, diffusion models [28] can be designed to share the same marginals $q(\boldsymbol{x}_t \mid \boldsymbol{x}_0)$ under equivalent noise schedules, but define a stochastic forward process whose discretized reverse takes the form of a Gaussian transition kernel $p_\theta(\boldsymbol{x}_s \mid \boldsymbol{x}_t) = \mathcal{N}(\boldsymbol{\mu}_\theta(\boldsymbol{x}_t, t, s),\ \sigma^2(t,s)\,\boldsymbol{I})$.

In both frameworks, generation quality depends on the number of discretization steps: a coarse numerical ODE step can poorly approximate the learned flow, while diffusion models assume the reverse conditional is Gaussian. With many steps these approximations are accurate; with few steps each transition must cover a large interval, and the true mapping from $\boldsymbol{x}_t$ to $\boldsymbol{x}_s$ becomes too complex for either a linear or Gaussian model to capture. To formalize and address this limitation, we adopt a stochastic trajectory framework that makes the per-step distribution an explicit modeling target.

2.2 Stochastic Trajectories and the Gaussian Bottleneck

Given a timestep schedule $0 = t_0 < t_1 < \cdots < t_T = 1$, we construct a Markovian forward trajectory that satisfies the marginal constraint in Eq. (1) at every step. For any two consecutive timesteps $s < t$ in the schedule, the forward transition is:

$$\boldsymbol{x}_t = \alpha_{s,t}\,\boldsymbol{x}_s + \sigma_{s,t}\,\boldsymbol{\epsilon}, \quad \alpha_{s,t} = \frac{1-t}{1-s}, \quad \sigma_{s,t} = \sqrt{t^2 - \alpha_{s,t}^2\,s^2}, \tag{3}$$

where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. Applying this transition sequentially yields a correlated stochastic path $(\boldsymbol{x}_{t_0}, \boldsymbol{x}_{t_1}, \ldots, \boldsymbol{x}_{t_T})$ from near-clean to near-noise, with each point marginally distributed as $q(\boldsymbol{x}_t \mid \boldsymbol{x}_0)$. The Markovian structure defines a tractable joint distribution over the trajectory whose reverse conditionals $q(\boldsymbol{x}_s \mid \boldsymbol{x}_t, \boldsymbol{x}_0)$ are Gaussian with known mean and variance.

The Gaussian approximation. Standard Gaussian diffusion samplers approximate the stochastic reverse conditional $p(\boldsymbol{x}_s \mid \boldsymbol{x}_t)$ with a single Gaussian $\mathcal{N}(\boldsymbol{\mu}_\theta(\boldsymbol{x}_t), \sigma^2\boldsymbol{I})$. For the chosen linear Gaussian forward process, the posterior conditioned on the clean image, $p(\boldsymbol{x}_s \mid \boldsymbol{x}_t, \boldsymbol{x}_0)$, is Gaussian. A deterministic probability-flow ODE has the same one-time marginals but a different coupling between times. However, the marginal reverse conditional integrates over all possible clean images:

$$p(\boldsymbol{x}_s \mid \boldsymbol{x}_t) = \int p(\boldsymbol{x}_s \mid \boldsymbol{x}_t, \boldsymbol{x}_0)\,p(\boldsymbol{x}_0 \mid \boldsymbol{x}_t)\,\mathrm{d}\boldsymbol{x}_0. \tag{4}$$

Since $p(\boldsymbol{x}_0 \mid \boldsymbol{x}_t)$ is complex and potentially multimodal over natural images, the marginal $p(\boldsymbol{x}_s \mid \boldsymbol{x}_t)$ is a mixture of Gaussians that a single Gaussian cannot capture. When the number of steps is small, each transition spans a large interval and the approximation error becomes severe.

2.3 Normalizing Flows

Normalizing flows [15; 42; 16; 31] learn an invertible mapping $f_\theta : \mathbb{R}^D \to \mathbb{R}^D$ between data $\boldsymbol{x}$ and a latent $\boldsymbol{z} = f_\theta(\boldsymbol{x})$ drawn from a simple prior $p_0(\boldsymbol{z}) = \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. The exact log-likelihood is given by the change-of-variables formula:

$$\log p(\boldsymbol{x}) = \log p_0\big(f_\theta(\boldsymbol{x})\big) + \log\big|\det J_{f_\theta}(\boldsymbol{x})\big|. \tag{5}$$

A common design is the autoregressive flow [32; 39], which transforms each element conditioned on all preceding elements via affine (NVP) coupling [16], yielding a tractable triangular Jacobian.

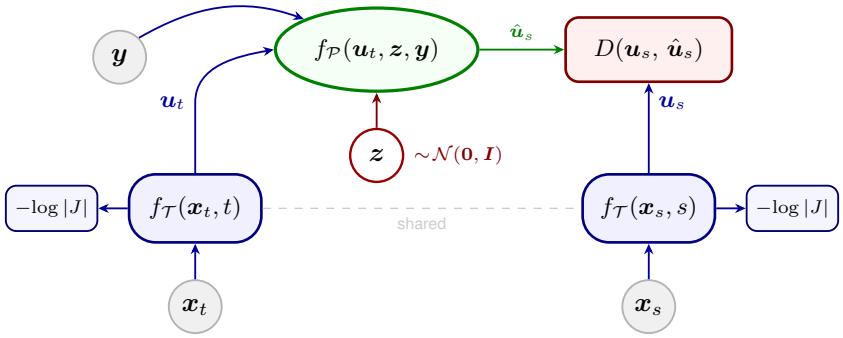


Figure 3: **NTM overview.** Shared transporter $f_{\mathcal{T}}$ maps $\boldsymbol{x}_t, \boldsymbol{x}_s$ to representations $\boldsymbol{u}_t, \boldsymbol{u}_s$ with a tractable Jacobian. The predictor $f_{\mathcal{P}}$ takes $\boldsymbol{u}_t$ and latent $\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ to produce $\hat{\boldsymbol{u}}_s$. D measures the distance between the prediction and the target at distribution level.

TarFlow [56] parameterizes the affine coupling with a causal Transformer: each spatial token $\boldsymbol{x}_n$ is transformed conditioned on all preceding tokens $\boldsymbol{x}_{<n}$ via a self-exclusive causal mask:

$$\boldsymbol{z}_n = \frac{\boldsymbol{x}_n - \boldsymbol{\mu}_\theta(\boldsymbol{x}_{<n})}{\boldsymbol{\sigma}_\theta(\boldsymbol{x}_{<n})}, \quad \log\left|\det J\right| = -\sum_n \log \boldsymbol{\sigma}_\theta^{(n)}, \tag{6}$$

where $\boldsymbol{\sigma}_\theta > 0$ (scale) and $\boldsymbol{\mu}_\theta$ (shift) are predicted from preceding tokens. This allows normalizing flows to scale competitively for high-resolution image generation. STARFlow [25] further introduces a *deep-shallow* architecture: a single deep autoregressive flow block with many Transformer layers captures most of the model capacity, followed by a few lightweight shallow blocks with alternating scan directions (e.g., left-to-right and right-to-left) that refine spatial details. This deep-shallow design, extended to video in STARFlow-V [24], forms the architectural foundation of NTM.

3 Normalizing Trajectory Models

We present Normalizing Trajectory Models (NTM), a generative framework that models the full conditional distribution $p(\boldsymbol{x}_s \mid \boldsymbol{x}_t)$ at each denoising step as a normalizing flow with exact log-likelihood (§ 3.1). NTM can be trained from scratch (§ 3.2), finetuned from pretrained diffusion or flow-matching models (§ 3.3), and accelerated to fast generation via a learned denoiser (§ 3.4).

3.1 Model Formulation

As discussed in § 2.2, modeling $p(\boldsymbol{x}_s \mid \boldsymbol{x}_t)$ with a Gaussian formulation is fundamentally limited: the true reverse conditional is generally non-Gaussian because it marginalizes over all clean images consistent with $\boldsymbol{x}_t$. We seek a more expressive family that provides exact likelihood for stable training, while remaining structurally close to the diffusion framework to preserve its scalability.

NTM models $p(\boldsymbol{x}_s \mid \boldsymbol{x}_t)$ by learning to predict in a latent space where the conditional distribution is simple enough to be modeled by Gaussian. As shown in Fig. 3, a shared transporter $f_{\mathcal{T}}$ maps both $\boldsymbol{x}_s$ and $\boldsymbol{x}_t$ to a latent u-space, and a stochastic predictor $f_{\mathcal{P}}$ generates $\hat{\boldsymbol{u}}_s$ from the noisier representation $\boldsymbol{u}_t$ and a latent variable $\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, optionally conditioned on $\boldsymbol{y}$ (e.g., text or class label).

$$\hat{\boldsymbol{u}}_s = f_{\mathcal{P}}(\boldsymbol{u}_t, \boldsymbol{z}, \boldsymbol{y}), \quad \boldsymbol{u}_s = f_{\mathcal{T}}(\boldsymbol{x}_s, s), \quad \boldsymbol{u}_t = f_{\mathcal{T}}(\boldsymbol{x}_t, t). \tag{7}$$

The general training objective minimizes a distributional distance D between the prediction and the target, regularized by $R(f_{\mathcal{T}})$ to prevent representation collapse [22]:

$$\mathcal{L} = \mathbb{E}_{\boldsymbol{z}}\left[D(\boldsymbol{u}_s, \hat{\boldsymbol{u}}_s)\right] + R(f_{\mathcal{T}}). \tag{8}$$

Such objectives are common in self-supervised representation learning [22; 10; 5; 4], but are generally difficult to cast within a probabilistic framework for generative modeling. The key insight of NTM is that making $f_{\mathcal{T}}$ an *invertible*, same-dimensional transporter—rather than a compressive encoder—turns this representation-learning objective into exact log-likelihood optimization via the change-of-variables formula.

Specifically, we implement $f_{\mathcal{T}}$ as a stack of TarFlow blocks [56; 25] with spatial NVP coupling (Eq. (6)), and $f_{\mathcal{P}}$ as an affine map $\hat{\boldsymbol{u}}_s = \boldsymbol{\mu}_{\mathcal{P}}(\boldsymbol{u}_t, t, s, \boldsymbol{y}) + \boldsymbol{\sigma}_{\mathcal{P}}(\boldsymbol{u}_t, t, s, \boldsymbol{y}) \cdot \boldsymbol{z}$, which defines $p_{\mathcal{P}}(\boldsymbol{u}_s \mid \boldsymbol{u}_t, \boldsymbol{y}) = \mathcal{N}(\boldsymbol{\mu}_{\mathcal{P}}, \operatorname{diag}(\boldsymbol{\sigma}_{\mathcal{P}}^2))$. Under these choices, setting $D = -\log p_{\mathcal{P}}$ and $R = -\log|\det J_{f_{\mathcal{T}}}|$ recovers the exact negative log-likelihood of $p(\boldsymbol{x}_s \mid \boldsymbol{x}_t)$:

$$\mathcal{L}_{\text{NTM}} = -\log p(\boldsymbol{x}_s \mid \boldsymbol{x}_t) = -\log p_{\mathcal{P}}(\boldsymbol{u}_s \mid \boldsymbol{u}_t) - \log\left|\det J_{f_{\mathcal{T}}}\right|. \tag{9}$$

The composed mapping $\boldsymbol{x}_s \overset{f_{\mathcal{T}}}{\longleftrightarrow} \boldsymbol{u}_s \overset{f_{\mathcal{P}}}{\longleftrightarrow} \boldsymbol{z}$ forms a **normalizing flow** from $\boldsymbol{x}_s$ to $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$. By expanding over a trajectory of T steps, the NTM loss can be simplified as:

$$\mathcal{L}_{\text{NTM}} = \sum_{k=1}^{T}\left[\tfrac{1}{2}\|\boldsymbol{z}_k\|^2 + \sum_{n}\left(\log \boldsymbol{\sigma}_{\mathcal{P}}^{(k,n)} + \sum_{\ell}\log \boldsymbol{\sigma}_{\mathcal{T}}^{(k,\ell,n)}\right)\right], \tag{10}$$

where $\boldsymbol{\sigma}_{\mathcal{P}}^{(k,n)}$ is the predictor scale at step k and position n, and $\boldsymbol{\sigma}_{\mathcal{T}}^{(k,\ell,n)}$ is the scale from transporter block ℓ. This is the trajectory NLL up to Gaussian constants and the fixed terminal-prior term, which do not affect parameter optimization. The transition Jacobians are evaluated directly, without ODE integration or a stochastic trace estimator. Exactness applies to transition and joint-trajectory densities; the endpoint marginal requires integrating out intermediate states. The fixed forward process yields an ELBO for that marginal (§ G.1).

Connection to stochastic flow maps. NTM can also be viewed as a stochastic flow map implemented with conditional normalizing flows [29; 33]. A deterministic flow map sends a fixed input state to one output, whereas a stochastic map takes fresh randomness and represents a distribution of possible outcomes over a finite interval. In NTM, this distribution is precisely the coarse reverse conditional $p_\theta(\boldsymbol{x}_s \mid \boldsymbol{x}_t)$: the predictor samples a Gaussian in u-space, and the inverse transporter maps it to x-space. Stochasticity alone does not resolve the Gaussian bottleneck, since a Gaussian reverse step already injects noise. The nonlinear invertible transporter expands the noise-to-state mapping while retaining a normalized, tractable conditional density. Composing these finite-interval kernels defines the trajectory model in Eq. (11); the resulting joint density supplies the score used for refinement. This is a transition-kernel interpretation, without an imposed consistency constraint across arbitrary schedules or a shared Brownian-path solution. The explicit stochastic map and its relation to other constructions appear in § G.4.

3.2 Training from Scratch

Architecture. NTM adopts the deep-shallow architecture of STARFlow [25; 24], with a key modification to the deep block. The *predictor* ($f_{\mathcal{P}}$) is a deep Transformer that replaces STARFlow's spatial autoregressive flow with a full-attention network predicting the affine transition between consecutive trajectory states. It predicts $\boldsymbol{\mu}_{\mathcal{P}}(\boldsymbol{u}_t, t, s, \boldsymbol{y})$ and $\boldsymbol{\sigma}_{\mathcal{P}}(\boldsymbol{u}_t, t, s, \boldsymbol{y})$ for each denoising step. Despite its depth, the predictor processes all spatial positions in parallel, making it efficient at inference. The *transporter* ($f_{\mathcal{T}}$) consists of a few shallow TarFlow-style [56] causal autoregressive flow blocks with alternating scan directions. Although autoregressive by nature, each transporter block is lightweight and operates locally within a single denoising step without information leakage across timestep.

Training. Given a T-step schedule $t_{\min} = t_0 < t_1 < \cdots < t_T = 1$, we model the joint trajectory distribution as:

$$p(\boldsymbol{x}_{t_T}, \ldots, \boldsymbol{x}_{t_0}) = p(\boldsymbol{x}_{t_T}) \prod_{k=1}^{T} p(\boldsymbol{x}_{t_{k-1}} \mid \boldsymbol{x}_{t_k}). \tag{11}$$

At $t_T = 1$, we use the Gaussian prior $p(\boldsymbol{x}_{t_T}) = \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ and the identity representation $\boldsymbol{u}_{t_T} = \boldsymbol{x}_{t_T}$. The first learned predictor transition starts from this noise state; training learns the conditional factors. Given clean data $\boldsymbol{x}_0$, we construct a stochastic forward trajectory via Eq. (3) and train with either:

- *End-to-end*: compute the NTM loss (Eq. (10)) over all T conditional factors in the trajectory.
- *Pair-wise*: randomly sample a single consecutive pair (t, s) with $s < t$ per batch element.

In both modes, each batch element independently samples T from a predefined set (e.g., $\{4, 8, 16\}$), enabling a single model to generate with different step counts without retraining. For such cases, $f_{\mathcal{T}}$ takes T as an additional input to adapt to the local timestep spacing.

Sampling. Given a schedule $t_{\min} = s_0 < s_1 < \cdots < s_T \approx 1$, sampling proceeds from noise to data by inverting Eq. (7): the predictor runs sequentially over T steps, drawing $\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ and computing $\hat{\boldsymbol{u}}_s = \boldsymbol{\mu}_{\mathcal{P}}(\hat{\boldsymbol{u}}_t, t, s) + \boldsymbol{\sigma}_{\mathcal{P}}(\hat{\boldsymbol{u}}_t, t, s) \cdot \boldsymbol{z}$ at each step, where each output feeds into the next. After all T predictor steps, the transporter inverts the spatial mapping $\hat{\boldsymbol{x}}_0 = f_{\mathcal{T}}^{-1}(\hat{\boldsymbol{u}}_0)$ via sequential AR decoding to produce the final sample in x-space. Classifier-free guidance [27] is applied by interpolating the predictor's conditional and unconditional outputs [25].

Trajectory Score Denoising. Normalizing flows require data to be dense for likelihood training, while natural images often lie on low-dimensional manifolds; TarFlow addresses this by adding a small noise and applying score-based denoising at test time [56; 25]. In NTM, this extends naturally: the generated trajectory $\hat{\boldsymbol{x}} = (\hat{\boldsymbol{x}}_{t_0}, \ldots, \hat{\boldsymbol{x}}_{t_T})$ is inherently a noisy sequence from the Markov forward process, requiring no additional noise injection. However, unlike independent per-sample denoising, the trajectory elements are correlated across timesteps. The normalized trajectory model supplies $-\log p(\hat{\boldsymbol{x}})$, whose gradient gives the *joint* score. The terminal state is held fixed during denoising; the correction is applied only at $t_i < 1$. We exploit this to perform trajectory-level denoising:

$$\hat{\boldsymbol{x}}^{\text{den}} = \frac{1}{1-\boldsymbol{t}} \left(\hat{\boldsymbol{x}} - \boldsymbol{S} \cdot \nabla_{\hat{\boldsymbol{x}}} \mathcal{L}_{\text{NTM}} \right), \tag{12}$$

where $\boldsymbol{S}$ is the covariance matrix of the trajectory under the pre-defined forward process (Eq. (3)), with $[\boldsymbol{S}]_{ij} = \min(t_i, t_j)^2 (1 - \max(t_i, t_j)) / (1 - \min(t_i, t_j))$, and division by $(1-\boldsymbol{t})$ maps from the noisy domain to the clean domain. The final output is taken at $t_{\min}$.

3.3 Finetuning from Pretrained Models

NTM can also be initialized from a pretrained flow matching or diffusion models. Taking flow matching as an example, the pretrained backbone is trained to predict the velocity field in x-space given noisy input $\boldsymbol{x}_t$ and timestep t. Here, we reinterpret the prediction $\boldsymbol{v}$ and hidden states $\boldsymbol{h}$ from the input $(\boldsymbol{u}_t, t)$ in u-space. We can readily compute a predicted clean sample $\hat{\boldsymbol{u}}_0 = \boldsymbol{u}_t - t \cdot \boldsymbol{v}$ and derive the denoising posterior $\mathcal{N}(\boldsymbol{\mu}_{\text{post}}, \sigma_{\text{post}}^2 \boldsymbol{I})$ for the transition from t to s:

$$\boldsymbol{\mu}_{\text{post}} = A(t,s)\, \boldsymbol{u}_t + B(t,s)\, \hat{\boldsymbol{u}}_0, \qquad \sigma_{\text{post}} = C(t,s), \tag{13}$$

where A, B, and C are closed-form coefficients derived from the true reverse posterior of the Markovian forward process (§ 2.2; full derivation in § A.5). We initialize the predictor to match this posterior: $\boldsymbol{\mu}_{\mathcal{P}} = \boldsymbol{\mu}_{\text{post}}$, and learn a multiplicative scale correction via a zero-initialized projection:

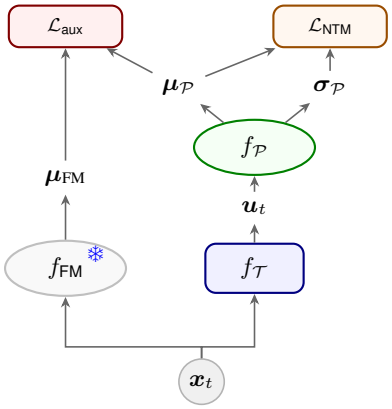


Figure 4: Finetuning: $\mathcal{L}_{\text{aux}}$ aligns $\boldsymbol{\mu}_{\mathcal{P}}$ with frozen $\boldsymbol{\mu}_{\text{FM}}$; $\mathcal{L}_{\text{NTM}}$ trains the full model.

$$\boldsymbol{\mu}_{\mathcal{P}} = \boldsymbol{\mu}_{\text{post}}, \qquad \boldsymbol{\sigma}_{\mathcal{P}} = \sigma_{\text{post}} \cdot \exp(\boldsymbol{\delta}_\sigma), \quad \boldsymbol{\delta}_\sigma = \text{proj}_{\text{out}}(\boldsymbol{h}), \tag{14}$$

where proj_{out} is initialized to zero so that $\boldsymbol{\sigma}_{\mathcal{P}} = \sigma_{\text{post}}$ at initialization. By further initializing the transporter as identity ($f_{\mathcal{T}} = \text{id}$), the full model starts as the pretrained Gaussian posterior in x-space. As training progresses, the NLL objective drives $f_{\mathcal{T}}$ to drift from identity and $\boldsymbol{\delta}_\sigma$ to depart from zero, jointly learning the non-Gaussian structure of $p(\boldsymbol{x}_s \mid \boldsymbol{x}_t)$.

Mean-alignment auxiliary loss. To prevent early divergence from the pretrained solution, we add an auxiliary loss that aligns NTM's learned shift $\boldsymbol{\mu}_{\mathcal{P}}$ with the denoising mean $\boldsymbol{\mu}_{\text{FM}}$ produced by a frozen copy of the pretrained backbone predicting directly from x-space:

$$\mathcal{L}_{\text{aux}} = \left\| \boldsymbol{\mu}_{\mathcal{P}} - \boldsymbol{\mu}_{\text{FM}} \right\|^2. \tag{15}$$

The total loss is $\mathcal{L} = \mathcal{L}_{\text{NTM}} + \lambda\, \mathcal{L}_{\text{aux}}$, where the reported finetuning runs use a fixed $\lambda = 2.5$. The teacher is frozen, while the student backbone, scale correction, and transporter remain trainable. Mean alignment is a soft penalty, including when the separate residual mean head is disabled. This auxiliary loss serves three purposes: (1) it encourages the model to remain close to the pretrained diffusion solution, preventing catastrophic drift; (2) $\boldsymbol{\mu}_{\text{FM}}$ itself defines a meaningful u-space—since

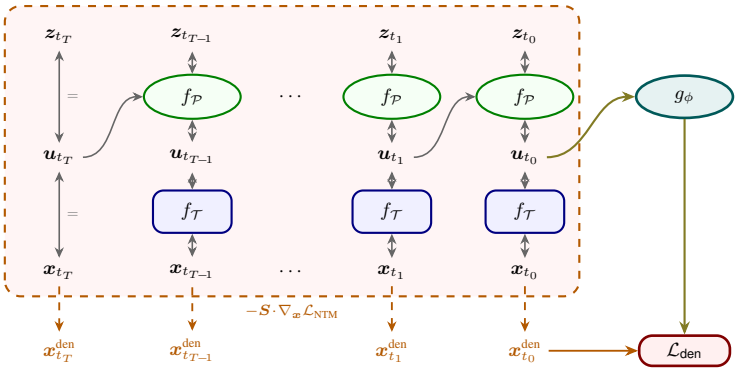


Figure 5: **Denoiser training via trajectory score denoising.** The frozen NTM (dashed box) computes the trajectory NLL and its gradient refines every position via $\boldsymbol{x}^{\text{den}} = \boldsymbol{x} - \boldsymbol{\Sigma}\nabla_{\boldsymbol{x}}\mathcal{L}_{\text{NTM}}$, producing denoised targets (orange). A denoiser g_ϕ learns to predict $\boldsymbol{x}^{\text{den}}_{t_0}$ directly from the $\boldsymbol{u}_{t_0}$.

it is a neural prediction of the next-step mean directly from $\boldsymbol{x}_t$, it is smooth and predictable, and $\mathcal{L}_{\text{aux}}$ ensures the transporter learns to connect these per-step predictions into a coherent trajectory; (3) because the transporter and predictor can move jointly, the model can optimize the NF loss without drifting from the pretrained quality.

3.4 Fast Generation via Learned Denoiser

The explicit refinement route runs T predictor transitions, inverts the transporter to recover the required trajectory states, and applies Eq. (12) using input gradients. Endpoint-only base sampling needs inversion only at the final state. Spatial inversion and test-time gradients add latency. To eliminate this cost, we can optionally train a lightweight denoiser network g_ϕ that amortizes the self-refinement into a single forward pass, following a similar distillation paradigm of NFM [6] and STARFlow-V [24]. The denoiser is a Transformer with non-causal attention that takes the predictor's output $\boldsymbol{u}_{t_0}$ at the cleanest level in u-space along with text embeddings $\boldsymbol{y}$, and directly outputs a denoised image $\hat{\boldsymbol{x}}_0^{\text{den}}$. The denoiser learns an MSE approximation to the refinement target from the final u-space state; exact recovery from that state alone is not guaranteed.

The denoiser can be post-trained after the main model converges, using MSE against score-based denoising targets derived from the frozen NTM model on real data trajectories (Eq. (12)):

$$\mathcal{L}_{\text{den}} = \left\| g_\phi(\boldsymbol{u}_{t_0}, \boldsymbol{y}) - \hat{\boldsymbol{x}}_0^{\text{den}} \right\|^2. \tag{16}$$

At inference, the new pipeline becomes: (1) run the predictor over T steps to produce $\boldsymbol{u}_{t_0}$, (2) run g_ϕ in a single forward pass to obtain $\hat{\boldsymbol{x}}_0$. All NTM results in Table 1 use this route: **four predictor transitions plus one denoiser pass**. It bypasses transporter inversion and test-time gradients. The denoiser is not constrained to be invertible and its Jacobian is not evaluated; exact output likelihood is therefore not claimed for the fast sampler.

Efficient composition in shared coordinates. The stochastic flow-map view also explains why intermediate x-space states need not be decoded during endpoint sampling. Writing $f_t(\boldsymbol{x}) = f_{\mathcal{T}}(\boldsymbol{x}, t)$ and $H_{t\to s}$ for the stochastic predictor, each map has the form $f_s^{-1} \circ H_{t\to s} \circ f_t$. When consecutive maps are composed on the same schedule, the intermediate $f_r \circ f_r^{-1}$ cancels. The predictor recurrence can therefore remain in u-space, processing spatial positions in parallel. The explicit route still needs endpoint inversion and additional trajectory reconstruction for score refinement; the learned denoiser amortizes these remaining operations into a forward readout. The speed thus follows from shared-coordinate composition and amortization, rather than stochasticity alone. We give the composition identity and its distinction from cross-schedule consistency in § G.4.

Table 1: **T2I Evaluation.** GenEval [20] overall score and DPG-Bench [30] percentage. Sampling counts describe different generation structures, not matched latency. A dash indicates that the sampling configuration for the cited score is not established here. NTM uses four predictor transitions plus one denoiser pass.

Type	Method	Sampling	GenEval↑	DPG↑
DM	SDXL [41]	—	0.55	74.65
	PixArt-α [11]	—	0.48	71.11
	SD3-Medium [17]	—	0.62	84.08
	FLUX.1-dev [7]	—	0.66	83.84
	Janus-Pro-7B [14]	—	0.80	84.19
	HiDream-I1-Full [26]	—	0.83	85.89
	Seedream 3.0 [9]	—	0.84	88.27
	Qwen-Image [51]	—	0.87	88.32
	Nucleus-Image [2]	—	0.87	88.79
NF	STARFlow [25]	256 AR positions	0.56	–
	NTM (scratch, 256^2)	$4+1$	0.82	79.64
	NTM (finetune, 512^2)	$4+1$	0.76	83.38

4 Experiments

4.1 Setup

Implementation. Text-to-image models are trained with AdamW in bfloat16 with FSDP on an internal corpus of approximately 70M licensed text-image pairs (including CC12M). The class-conditional experiments use the public ImageNet dataset. We consider two settings:

- *From scratch*: class-conditional ImageNet and text-to-image generation at 256×256 resolution with the latent space of FAE [18] ($16\times$ spatial compression, 32-dim latents), using Qwen-2.5-VL as the text encoder.
- *Finetuning*: initializing from a pretrained flow-matching backbone (FLUX.2-klein, 4B)[2] at 512×512 resolution with its native VAE latent space.

The transporter consists of 2 TarFlow-style blocks with 4 layers each and causal masks along alternating directions; the predictor is a 24-layer full-attention Transformer. Unless otherwise specified, text-to-image models use $T=4$ predictor transitions followed by one learned-denoiser pass. Training uses 10% CFG dropout. For finetuning, we apply the residual parameterization (§ 3.3) with the auxiliary loss ($\lambda=2.5$, MSE variant). Both settings use a batch size of 1024 on 64 H100 GPUs. Further details are in the Appendix.

Evaluation. We report compositional accuracy on GenEval [20] and DPG-Bench [30] for text-to-image generation. We additionally evaluate class-conditional generation on ImageNet 256×256 for fair comparison when training NTM from scratch (§ D.3).

4.2 From-Scratch Results

Text-to-image generation. Table 1 reports compositional accuracy on GenEval and DPG-Bench. NTM trained from scratch at 256×256 achieves **0.82** on GenEval and **79.64** on DPG-Bench with four predictor transitions and one denoiser pass. STARFlow [25] reports 0.56 GenEval using 256 spatial autoregressive positions. These are different generation structures, rather than matched latency measurements.

Class-conditional ImageNet. As a controlled comparison for the from-scratch setting, we evaluate on class-conditional ImageNet 256×256. NTM achieves 2.80 FID with 16 steps and 3.83 FID with 4 steps—close in image quality to STARFlow (FAE) at 2.67 FID with 256 spatial autoregressive positions (§ D.3). The base model is trained with exact transition NLL and no distribution-level losses (e.g., adversarial or perceptual); the denoiser is trained separately with MSE, supporting image-quality competitiveness among normalizing-flow models; these counts do not establish matched latency.

[2] https://huggingface.co/black-forest-labs/FLUX.2-klein-base-4B

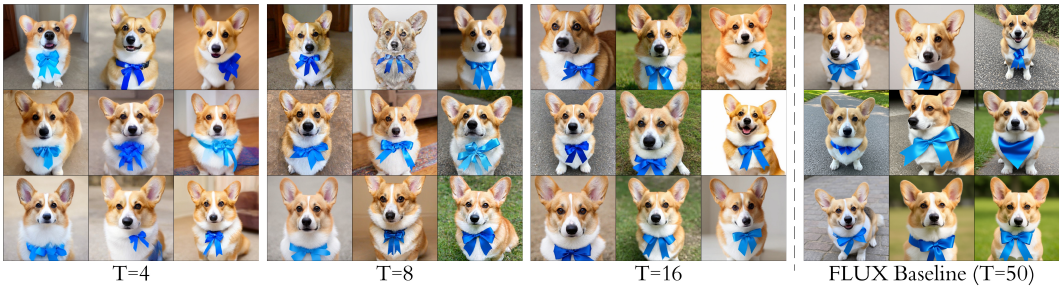
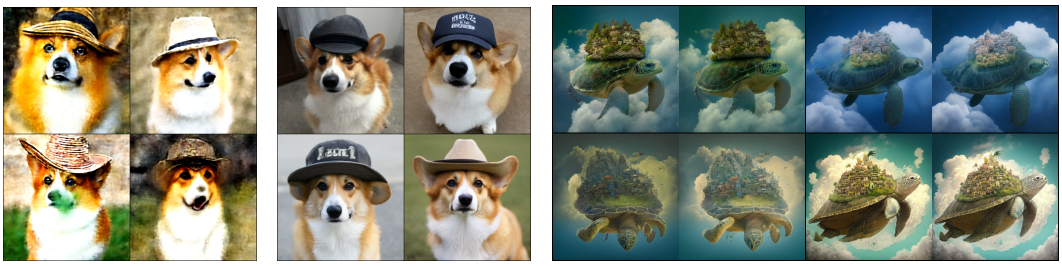


Figure 6: **Ablation: multi-trajectory training.** Comparison of the same NTM evaluated with T=4, T=8, T=16 denoising steps and the baseline FLUX (50 steps).



(a) Without aux loss (b) With aux loss (c) Trajectory score denoising *vs.* learned denoiser

Figure 7: **Ablations.** (a) Finetuning directly with the NF loss diverges. (b) Adding the mean-alignment loss (Eq. (15)) stabilizes training. (c) Comparison of denoising approaches.

4.3 Finetuning Results

Text-to-image generation. The finetuned variant at 512×512 achieves 0.76 on GenEval and 83.38 on DPG-Bench (Table 1), demonstrating that NTM can scale to higher resolutions via pretrained initialization. The position and attribute-binding sub-tasks remain challenging at this stage of finetuning, suggesting room for improvement with longer training or stronger pretrained backbones.

Score denoising vs. learned denoiser. Table 2 compares two inference strategies for the finetuned model: (i) transporter inversion followed by trajectory score denoising via Eq. (12), and (ii) the learned denoiser g_ϕ that amortizes the refinement into a single forward pass. The denoiser achieves **9.4**$\times$ speedup while maintaining high fidelity to the score-based refinement output (LPIPS 0.121 after decoding, relative to the explicit output), confirming that a single forward pass can effectively replace iterative backpropagation-based denoising.

Table 2: Explicit vs. learned correction (finetuned; B200, batch size 1).

Method	img/s ↑	LPIPS ↓
Full NF + Traj. denoise	0.20	—
Predictor + Denoiser	1.88	0.121

4.4 Ablation Studies

We conduct ablation studies on text-to-image generation to analyze the key design choices of NTM.

Multi-trajectory training (finetuned). Fig. 6 compares finetuned models trained with different trajectory lengths $T \in \{4, 8, 16\}$ against the baseline FLUX (50 steps). Longer trajectories provide finer-grained denoising steps, which can improve detail preservation at the cost of slower inference. We find that T=4 provides the best quality–speed trade-off for the finetuning setting.

Effect of the transporter (from scratch). As shown in Fig. 2, reducing flow matching to 4 steps without a transporter produces severely blurry outputs. The invertible mapping provides a latent space where the affine predictor becomes expressive, giving a sharp four-transition output in this qualitative comparison.

Auxiliary loss for finetuning. Fig. 7 ablates the mean-alignment auxiliary loss (Eq. (15)). Without the auxiliary loss ($\lambda = 0$), finetuning diverges early in the reported ablation. This establishes an optimization benefit of mean alignment, without isolating the cause of divergence. The auxiliary loss stabilizes training by anchoring the predictor mean to the pretrained velocity field.

4.5 Qualitative Results

Fig. 1 presents text-to-image samples from NTM using four predictor transitions plus one denoiser pass across both settings.

From scratch (256×256)**.** The from-scratch model demonstrates strong compositional generalization across multi-object scenes, fine-grained attribute control, and varied artistic styles despite being trained at moderate resolution.

Finetuned (512×512)**.** The finetuned model preserves the visual quality and prompt adherence of the pretrained FLUX backbone (which requires 50 steps) using four predictor transitions and one denoiser pass. Samples exhibit high-resolution detail, text rendering capability, and diverse artistic styles.

Limitations and architectural tradeoffs. The shallow transporter allocates capacity across coarse reverse transitions. Single-transition generation remains challenging; deeper or adaptive-depth transporters are possible future directions, with increased autoregressive decoding cost. We discuss this tradeoff in § G.

5 Related Work

Normalizing flows and coarse transitions. TarFlow [56] and STARFlow [25; 24] scale normalizing flows using autoregressive Transformers and deep–shallow blocks. NTM reuses these primitives for coarse reverse conditionals, rather than a marginal image density. DDGAN [52] learns non-Gaussian reverse transitions adversarially; UDPM [1], DiffRatio [13], and Transition Matching [46] explore alternative few-step formulations. Our distinction is the normalized conditional likelihood and its joint-trajectory refinement signal.

Stochastic flow maps. Deterministic flow maps [8; 19] learn finite-time transports. Diamond Maps [29] retain stochastic conditional sampling for reward alignment, while Neural Stochastic Flows [33] already use conditional normalizing flows for SDE transition laws. NTM can also be viewed in this transition-kernel sense, with conditional normalizing flows implementing its coarse reverse transitions. Strong stochastic flow maps [38] instead target Brownian-path-conditioned solution maps. Concurrent work explores conditional flows for corrected Boltzmann sampling [21] and invertible two-time pushforward maps [44]. Further distinctions and other related work appear in §§ G.4 and H.

6 Conclusion

We introduced Normalizing Trajectory Models (NTM), a framework that models coarse reverse conditionals as normalizing flows with exact transition and joint-trajectory density evaluation. The framework supports training from scratch and pretrained initialization. Its trajectory score supervises a fast sampler using four predictor transitions and one learned-denoiser pass. Future work includes distribution-level post-training, higher resolutions, and architectures for fewer transitions.

References

[1] Shady Abu-Hussein and Raja Giryes. UDPM: Upsampling diffusion probabilistic models. In *Advances in Neural Information Processing Systems*, 2024.

[2] Chandan Akiti et al. Nucleus-image: Sparse moe for image generation. *arXiv preprint arXiv:2604.12163*, 2026.

[3] Michael S Albergo, Nicholas M Boffi, and Eric Vanden-Eijnden. Stochastic interpolants: A unifying framework for flows and diffusions. *Journal of Machine Learning Research*, 26(209): 1–80, 2025. URL https://www.jmlr.org/papers/v26/23-1605.html.

[4] Mahmoud Assran, Quentin Duval, Ishan Misra, Piotr Bojanowski, Pascal Vincent, Michael Rabbat, Yann LeCun, and Nicolas Ballas. Self-supervised learning from images with a joint-embedding predictive architecture. *arXiv preprint arXiv:2301.08243*, 2023.

[5] Adrien Bardes, Jean Ponce, and Yann LeCun. Vicreg: Variance-invariance-covariance regularization for self-supervised learning. *arXiv preprint arXiv:2105.04906*, 2022.

[6] David Berthelot, Tianrong Chen, Jiatao Gu, Marco Cuturi, Laurent Dinh, Bhavik Chandna, Michal Klein, Josh Susskind, and Shuangfei Zhai. The coupling within: Flow matching via distilled normalizing flows. *arXiv preprint arXiv:2603.09014*, 2026.

[7] Black Forest Labs. Flux.1. Technical report / model release, 2024.

[8] Nicholas M Boffi, Michael S Albergo, and Eric Vanden-Eijnden. How to build a consistency model: Learning flow maps via self-distillation. *arXiv preprint arXiv:2505.18825*, 2025.

[9] ByteDance Seed Team. Seedream 3.0. Technical report / model release, 2025.

[10] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. *arXiv preprint arXiv:2104.14294*, 2021.

[11] Junsong Chen, Jincheng Yu, Chongjian Ge, Lewei Yao, Enze Xie, Yue Wu, Zhongdao Wang, James Kwok, Ping Luo, Huchuan Lu, and Zhenguo Li. Pixart-α: Fast training of diffusion transformer for photorealistic text-to-image synthesis. In *International Conference on Learning Representations*, 2024.

[12] Tianrong Chen, Jiatao Gu, David Berthelot, Joshua Susskind, and Shuangfei Zhai. Normalizing flows with iterative denoising. *arXiv preprint arXiv:2604.20041*, 2026.

[13] Wenlin Chen, Mingtian Zhang, Jiajun He, Zijing Ou, José Miguel Hernández-Lobato, Bernhard Schölkopf, and David Barber. DiffRatio: Training one-step diffusion models without teacher supervision. *arXiv preprint arXiv:2502.08005*, 2026.

[14] Xiaokang Chen, Zhiyu Wu, Xingchao Liu, Zizheng Pan, Wen Liu, Zhenda Xie, Xingkai Yu, and Chong Ruan. Janus-pro: Unified multimodal understanding and generation with data and model scaling. *arXiv preprint arXiv:2501.17811*, 2025.

[15] Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

[16] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

[17] Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, Dustin Podell, Tim Dockhorn, Zion English, Kyle Lacey, Alex Goodwin, Yannik Marek, and Robin Rombach. Scaling rectified flow transformers for high-resolution image synthesis. In *International Conference on Machine Learning*, 2024.

[18] Yuan Gao, Chen Chen, Tianrong Chen, and Jiatao Gu. One layer is enough: Adapting pretrained visual encoders for image generation. *arXiv preprint arXiv:2512.07829*, 2025.

[19] Zhengyang Geng, Mingyang Deng, Xingjian Bai, J Zico Kolter, and Kaiming He. Mean flows for one-step generative modeling. *arXiv preprint arXiv:2505.13447*, 2025.

[20] Dhruba Ghosh, Hannaneh Hajishirzi, and Ludwig Schmidt. Geneval: An object-focused framework for evaluating text-to-image alignment. In *Advances in Neural Information Processing Systems*, 2023.

[21] Louis Grenioux, RuiKang OuYang, and Luhuan Wu. Bridging stochastic flow maps and boltzmann generators with normalizing flows. *arXiv preprint arXiv:2609.31978*, 2026.

[22] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, et al. Bootstrap your own latent-a new approach to self-supervised learning. *Advances in neural information processing systems*, 33:21271–21284, 2020.

[23] Jiatao Gu, Yuyang Wang, Yizhe Zhang, Qihang Zhang, Dinghuai Zhang, Navdeep Jaitly, Josh Susskind, and Shuangfei Zhai. Dart: Denoising autoregressive transformer for scalable text-to-image generation. *arXiv preprint arXiv:2410.08159*, 2024.

[24] Jiatao Gu, Ying Shen, Tianrong Chen, Laurent Dinh, Yuyang Wang, Miguel Angel Bautista, David Berthelot, Josh Susskind, and Shuangfei Zhai. Starflow-v: End-to-end video generative modeling with normalizing flow. *arXiv preprint arXiv:2511.20462*, 2025.

[25] Jiatao Gu, Tianrong Chen, David Berthelot, Huangjie Zheng, Yuyang Wang, Ruixiang Zhang, Laurent Dinh, Miguel Angel Bautista, Joshua Susskind, and Shuangfei Zhai. Starflow: Scaling latent normalizing flows for high-resolution image synthesis. *Advances in Neural Information Processing Systems*, 38:120986–121022, 2026.

[26] HiDream.ai. Hidream-i1. Technical report / model release, 2025.

[27] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. In *NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications*, 2022.

[28] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

[29] Peter Holderrieth, Douglas Chen, Luca Eyring, Ishin Shah, Giri Anantharaman, Yutong He, Zeynep Akata, Tommi Jaakkola, Nicholas Matthew Boffi, and Max Simchowitz. Diamond maps: Efficient reward alignment via stochastic flow maps. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306, pages 43401–43435. PMLR, 2026. URL https://proceedings.mlr.press/v306/holderrieth26a.html.

[30] Xiwei Hu, Rui Wang, Yixiao Fang, Bin Fu, Pei Cheng, and Gang Yu. Ella: Equip diffusion models with llm for enhanced semantic alignment. *arXiv preprint arXiv:2403.05135*, 2024.

[31] Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. *Advances in neural information processing systems*, 31, 2018.

[32] Durk P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. *Advances in neural information processing systems*, 29, 2016.

[33] Naoki Kiyohara, Edward Johns, and Yingzhen Li. Neural stochastic flows: Solver-free modelling and inference for SDE solutions. In *Advances in Neural Information Processing Systems*, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/18abbeef8cfe9203fdf9053c9c4fe191-Abstract-Conference.html.

[34] Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. Flow matching for generative modeling. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=PqvMRDCJT9t.

[35] Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. In *International Conference on Learning Representations*, 2023.

[36] Simian Luo, Yiqin Tan, Longbo Huang, Jian Li, and Hang Zhao. Latent consistency models: Synthesizing high-resolution images with few-step inference. *arXiv preprint arXiv:2310.04378*, 2023.

[37] Nanye Ma, Mark Goldstein, Michael S Albergo, Nicholas M Boffi, Eric Vanden-Eijnden, and Saining Xie. Sit: Exploring flow and diffusion-based generative models with scalable interpolant transformers. In *European Conference on Computer Vision*, 2024.

[38] Sam McCallum, Zander W. Blasingame, Timothy Herschell, Niklas Rindtorff, Alexander Tong, and James Foster. Strong stochastic flow maps. *arXiv preprint arXiv:2606.01086*, 2026.

[39] George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation. *Advances in neural information processing systems*, 30, 2017.

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

[41] Dustin Podell, Zion English, Kyle Lacey, Andreas Blattmann, Tim Dockhorn, Jonas Müller, Joe Penna, and Robin Rombach. Sdxl: Improving latent diffusion models for high-resolution image synthesis. In *International Conference on Learning Representations*, 2024.

[42] Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International conference on machine learning*, pages 1530–1538. PMLR, 2015.

[43] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022.

[44] Francesco M. Ruscio, Timothy James Hitge, Fred Zhangzhi Peng, Danyal Rehman, Zander W. Blasingame, Nicholas M. Boffi, Michael M. Bronstein, Alexander Tong, and Avishek Joey Bose. Normalizing flow maps. *arXiv preprint arXiv:2610.22711*, 2026.

[45] Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. In *International Conference on Learning Representations*, 2022.

[46] Neta Shaul, Uriel Singer, Itai Gat, and Yaron Lipman. Transition matching: Scalable and flexible generative modeling. *arXiv preprint arXiv:2506.23589*, 2025.

[47] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*, 2021.

[48] Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models. In *International Conference on Machine Learning*, 2023.

[49] Peize Sun, Yi Jiang, Shoufa Chen, Shilong Zhang, Bingyue Peng, Ping Luo, and Zehuan Yuan. Autoregressive model beats diffusion: Llama for scalable image generation. *arXiv preprint arXiv:2406.06525*, 2024.

[50] Keyu Tian, Yi Jiang, Zehuan Yuan, Bingyue Peng, and Liwei Wang. Visual autoregressive modeling: Scalable image generation via next-scale prediction. *arXiv preprint arXiv:2404.02905*, 2024.

[51] Chenfei Wu et al. Qwen-image technical report. *arXiv preprint*, 2025.

[52] Zhisheng Xiao, Karsten Kreis, and Arash Vahdat. Tackling the generative learning trilemma with denoising diffusion GANs. In *International Conference on Learning Representations*, 2022.

[53] Jiawei Yang, Zhengyang Geng, Xuan Ju, Yonglong Tian, and Yue Wang. Representation fréchet loss for visual generation. *arXiv preprint arXiv:2604.28190*, 2026.

[54] Tianwei Yin, Michaël Gharbi, Taesung Park, Richard Zhang, Eli Shechtman, Frédo Durand, and William T Freeman. Improved distribution matching distillation for fast image synthesis. *Advances in Neural Information Processing Systems*, 2024.

[55] Tianwei Yin, Michaël Gharbi, Richard Zhang, Eli Shechtman, Frédo Durand, William T Freeman, and Taesung Park. One-step diffusion with distribution matching distillation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

[56] Shuangfei Zhai, Ruixiang ZHANG, Preetum Nakkiran, David Berthelot, Jiatao Gu, Huangjie Zheng, Tianrong Chen, Miguel Ángel Bautista, Navdeep Jaitly, and Joshua M Susskind. Normalizing flows are capable generative models. In *Forty-second International Conference on Machine Learning*, 2025.

[57] Qinsheng Zhang and Yongxin Chen. Diffusion normalizing flow. In *Advances in Neural Information Processing Systems*, 2021.

A Theoretical Analysis

A.1 NTM as a Conditional Normalizing Flow

We establish that each reverse transition in NTM defines a valid conditional normalizing flow with exact log-likelihood. Consider a single denoising step from timestep t to s ($s < t$). NTM maps the clean-side sample $\boldsymbol{x}_s$ to a latent $\boldsymbol{z}$ via the composition of two invertible transformations:

1. **Transporter** (spatial autoregressive flow): $\boldsymbol{u}_s = f_{\mathcal{T}}(\boldsymbol{x}_s)$, an invertible mapping from x-space to u-space via a stack of L TarFlow-style [56] causal AR blocks with alternating scan directions. Each block ℓ applies an elementwise affine coupling $\boldsymbol{u}_n^{(\ell)} = (\boldsymbol{u}_n^{(\ell-1)} - \boldsymbol{\mu}^{(\ell)}(\boldsymbol{u}_{<n}^{(\ell-1)}))/\boldsymbol{\sigma}^{(\ell)}(\boldsymbol{u}_{<n}^{(\ell-1)}))$ with a triangular Jacobian.
2. **Predictor** (trajectory-level affine coupling): $\boldsymbol{z} = (\boldsymbol{u}_s - \boldsymbol{\mu}_{\mathcal{P}}(\boldsymbol{u}_t, t, s))/\boldsymbol{\sigma}_{\mathcal{P}}(\boldsymbol{u}_t, t, s)$, an affine coupling in u-space conditioned on the noisier representation $\boldsymbol{u}_t$.

The composition $\boldsymbol{z} = f_{\mathcal{P}}(f_{\mathcal{T}}(\boldsymbol{x}_s); \boldsymbol{u}_t, t, s)$ is invertible (both components are), and the exact log-likelihood follows from the change-of-variables formula:

$$\log p(\boldsymbol{x}_s \mid \boldsymbol{x}_t) = \underbrace{\log p_0(\boldsymbol{z})}_{\text{Gaussian prior}} + \underbrace{\log \left|\det J_{f_{\mathcal{P}}}\right|}_{\text{predictor}} + \underbrace{\sum_{\ell=1}^{L} \log \left|\det J_{f_{\mathcal{T}}^{(\ell)}}\right|}_{\text{transporter}}. \tag{17}$$

Since the predictor is a diagonal affine coupling, $\log|\det J_{f_{\mathcal{P}}}| = -\sum_n \log \boldsymbol{\sigma}_{\mathcal{P}}^{(n)}$. Each transporter block has a triangular Jacobian from the autoregressive structure. Expanding the Gaussian prior term $\log p_0(\boldsymbol{z}) = -\frac{1}{2}\|\boldsymbol{z}\|^2 + \text{const}$ recovers Eq. (9) in the main text.

Relation to STARFlow. STARFlow [25] is a normalizing flow that models the *marginal* image distribution $p(\boldsymbol{x})$ using the same deep-shallow architectural building blocks. NTM applies these blocks to a fundamentally different object: the *conditional* distribution $p(\boldsymbol{x}_s \mid \boldsymbol{x}_t)$ at each denoising step. This has two consequences. First, STARFlow uses a deep spatially autoregressive block, whereas the NTM predictor uses full attention over spatial positions to parameterize a reverse transition conditioned on the noisier state. Consecutive training pairs can be evaluated in parallel, while reverse transitions remain sequential at sampling time. Second, because the conditional $p(\boldsymbol{x}_s \mid \boldsymbol{x}_t)$ is simpler than the marginal $p(\boldsymbol{x})$—conditioning on $\boldsymbol{x}_t$ already constrains the space of plausible images—NTM requires fewer transporter blocks per step to achieve comparable expressiveness.

A.2 Decomposition: Gaussian Denoising + Spatial Flow

NTM cleanly decomposes into two complementary components: a Gaussian denoising model in u-space (the predictor) and a non-Gaussian spatial transformation (the transporter). We formalize this decomposition and show that without the transporter, NTM reduces to the diagonal-Gaussian reverse-transition family.

Predictor alone = Gaussian denoising. Suppose the transporter is absent, i.e., $f_{\mathcal{T}} = \text{id}$ so that $\boldsymbol{u}_s = \boldsymbol{x}_s$. The latent variable becomes $\boldsymbol{z} = (\boldsymbol{x}_s - \boldsymbol{\mu}_{\mathcal{P}}(\boldsymbol{x}_t, t, s))/\boldsymbol{\sigma}_{\mathcal{P}}(\boldsymbol{x}_t, t, s)$, and the conditional distribution implied by the predictor is:

$$p(\boldsymbol{x}_s \mid \boldsymbol{x}_t) = \mathcal{N}\big(\boldsymbol{\mu}_{\mathcal{P}}(\boldsymbol{x}_t, t, s),\ \text{diag}(\boldsymbol{\sigma}_{\mathcal{P}}^2)\big). \tag{18}$$

This is a diagonal Gaussian, as in a conventional Gaussian reverse-diffusion kernel. A flow-matching ODE instead defines deterministic transport; its velocity prediction can be used to parameterize a Gaussian denoising posterior, as in our finetuning initialization. The NTM loss in this case reduces to:

$$\mathcal{L} = \sum_k \Big[\frac{1}{2}\Big\|\frac{\boldsymbol{x}_{s_k} - \boldsymbol{\mu}_{\mathcal{P}}^{(k)}}{\boldsymbol{\sigma}_{\mathcal{P}}^{(k)}}\Big\|^2 + \sum_n \log \boldsymbol{\sigma}_{\mathcal{P}}^{(k,n)}\Big], \tag{19}$$

which is the negative log-likelihood of a heteroscedastic Gaussian regression. If $\boldsymbol{\sigma}_{\mathcal{P}}$ is further held fixed, minimizing over $\boldsymbol{\mu}_{\mathcal{P}}$ yields a weighted MSE objective for the reverse-transition mean. This connects the Gaussian special case to denoising regression, without identifying a stochastic transition with deterministic ODE integration.

Transporter adds non-Gaussian expressiveness. The shallow autoregressive blocks introduce a *nonlinear*, invertible change of coordinates $\boldsymbol{u}_s = f_{\mathcal{T}}(\boldsymbol{x}_s)$ before the Gaussian coupling is applied. Even though the predictor still applies a Gaussian (affine) coupling in u-space, the *implied distribution in x-space* is non-Gaussian because the inverse $\boldsymbol{x}_s = f_{\mathcal{T}}^{-1}(\boldsymbol{u}_s)$ is a nonlinear transformation of a Gaussian. Formally:

$$p(\boldsymbol{x}_s \mid \boldsymbol{x}_t) = \mathcal{N}\big(f_{\mathcal{T}}(\boldsymbol{x}_s);\, \boldsymbol{\mu}_{\mathcal{P}},\, \mathrm{diag}(\boldsymbol{\sigma}_{\mathcal{P}}^2)\big) \cdot \big|\det J_{f_{\mathcal{T}}}(\boldsymbol{x}_s)\big|. \tag{20}$$

The Jacobian determinant $|\det J_{f_{\mathcal{T}}}|$ reweights the density to account for the nonlinear warping, allowing the model to represent multimodal, heavy-tailed, or skewed distributions in x-space even though u-space remains Gaussian.

Division of labor. In practice, the predictor captures the bulk of the denoising signal—predicting a good mean $\boldsymbol{\mu}_{\mathcal{P}}$ and an appropriate scale $\boldsymbol{\sigma}_{\mathcal{P}}$ in u-space—while the transporter learns the residual non-Gaussian structure via their spatial autoregressive coupling. This division is efficient: the predictor uses a large Transformer backbone and concentrates most parameters on cross-timestep reasoning, while each of the two transporter blocks has four Transformer layers and handles spatial dependencies within one reverse interval.

A.3 Effect of the FM Auxiliary Loss

When finetuning from a pretrained flow-matching model, we optimize

$$\mathcal{L} = \mathcal{L}_{\text{NTM}} + \lambda \mathcal{L}_{\text{aux}}, \qquad \mathcal{L}_{\text{aux}} = \sum_k \big\|\boldsymbol{\mu}_{\mathcal{P}}^{(k)} - \boldsymbol{\mu}_{\text{FM}}^{(k)}\big\|^2, \tag{21}$$

where a frozen pretrained teacher supplies $\boldsymbol{\mu}_{\text{FM}}$, and the reported checkpoints use a fixed $\lambda = 2.5$ throughout training. The teacher is frozen, but the student backbone producing $\boldsymbol{\mu}_{\mathcal{P}}$ remains trainable under both losses. The transporter and predictor scale are learned as well.

Optimization role. The NLL jointly optimizes the predictor and the transporter, allowing them to change their representation together. In the reported ablation, removing mean alignment ($\lambda = 0$) leads to early divergence, whereas the fixed-λ recipe trains stably. This comparison establishes the practical role of the regularizer in the evaluated setup; it does not isolate every possible cause of instability or establish that NLL-only finetuning is impossible.

Soft regularization rather than a fixed conditional. At finite λ, mean alignment penalizes departure from a reference prediction but does not fix the student mean to that prediction. In particular, `no_delta_mean=1` disables only the separate additive residual mean head. The velocity prediction of the trainable student backbone still changes $\boldsymbol{\mu}_{\text{post}}$ and hence $\boldsymbol{\mu}_{\mathcal{P}}$. The learned scale and nonlinear transporter provide further modeling freedom. Thus the model is initialized from a Gaussian reverse transition but is not constrained to remain that pretrained transition.

Large-weight limit. Formally taking $\lambda \to \infty$ would constrain the student mean to the teacher target on the training inputs, when such a constraint is feasible. This alone does not freeze the learned scale or remove the nonlinear transporter, so it does not make the full conditional equivalent to a fixed pretrained Gaussian kernel. Setting the transporter to identity recovers a Gaussian conditional in the modeled x-space; matching a particular pretrained kernel further requires matching its mean and scale.

A.4 Forward Transition Preserves Marginals

Proposition 1. *Let $q(\boldsymbol{x}_t \mid \boldsymbol{x}_0) = \mathcal{N}((1-t)\boldsymbol{x}_0, t^2\boldsymbol{I})$ and define the forward transition from time s to t ($s < t$) as:*

$$\boldsymbol{x}_t = \alpha_{s,t}\, \boldsymbol{x}_s + \sigma_{s,t}\, \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}), \tag{22}$$

with $\alpha_{s,t} = (1-t)/(1-s)$ and $\sigma_{s,t} = \sqrt{t^2 - \alpha_{s,t}^2 s^2}$. If $\boldsymbol{x}_s \sim q(\boldsymbol{x}_s \mid \boldsymbol{x}_0)$, then $\boldsymbol{x}_t \sim q(\boldsymbol{x}_t \mid \boldsymbol{x}_0)$.

Proof. Since $\boldsymbol{x}_s \sim \mathcal{N}((1-s)\boldsymbol{x}_0,\ s^2\boldsymbol{I})$, the transformed variable $\boldsymbol{x}_t = \alpha_{s,t}\boldsymbol{x}_s + \sigma_{s,t}\boldsymbol{\epsilon}$ is Gaussian with:

$$\mathbb{E}[\boldsymbol{x}_t \mid \boldsymbol{x}_0] = \alpha_{s,t}\,(1-s)\boldsymbol{x}_0 = \frac{1-t}{1-s}(1-s)\boldsymbol{x}_0 = (1-t)\boldsymbol{x}_0, \tag{23}$$

$$\mathrm{Var}[\boldsymbol{x}_t \mid \boldsymbol{x}_0] = \alpha_{s,t}^2 s^2 + \sigma_{s,t}^2 = \alpha_{s,t}^2 s^2 + t^2 - \alpha_{s,t}^2 s^2 = t^2. \tag{24}$$

Hence $\boldsymbol{x}_t \sim \mathcal{N}((1-t)\boldsymbol{x}_0, t^2\boldsymbol{I}) = q(\boldsymbol{x}_t \mid \boldsymbol{x}_0)$. □

A.5 Reverse Posterior Coefficients

We derive the closed-form expressions for the coefficients $A(t,s)$, $B(t,s)$, and $C(t,s)$ in the reverse Gaussian posterior $q(\boldsymbol{x}_s \mid \boldsymbol{x}_t, \boldsymbol{x}_0)$, used in the finetuning parameterization (Eq. (13)).

Proposition 2. *Under the forward process $q(\boldsymbol{x}_t \mid \boldsymbol{x}_0) = \mathcal{N}((1-t)\boldsymbol{x}_0, t^2\boldsymbol{I})$ with Markov transition (Eq. (3)), the reverse posterior is:*

$$q(\boldsymbol{x}_s \mid \boldsymbol{x}_t, \boldsymbol{x}_0) = \mathcal{N}\big(A(t,s)\,\boldsymbol{x}_t + B(t,s)\,\boldsymbol{x}_0,\ \ C(t,s)^2\,\boldsymbol{I}\big), \tag{25}$$

where

$$A(t,s) = \frac{s^2(1-t)}{t^2(1-s)}, \tag{26}$$

$$B(t,s) = \frac{(t-s)(t+s-2ts)}{t^2(1-s)}, \tag{27}$$

$$C(t,s)^2 = \frac{s^2(t-s)(t+s-2ts)}{t^2(1-s)^2}. \tag{28}$$

Proof. By Bayes' rule, $q(\boldsymbol{x}_s \mid \boldsymbol{x}_t, \boldsymbol{x}_0) \propto q(\boldsymbol{x}_t \mid \boldsymbol{x}_s)\, q(\boldsymbol{x}_s \mid \boldsymbol{x}_0)$, where:

$$q(\boldsymbol{x}_s \mid \boldsymbol{x}_0) = \mathcal{N}\big((1-s)\boldsymbol{x}_0,\ s^2\boldsymbol{I}\big), \tag{29}$$

$$q(\boldsymbol{x}_t \mid \boldsymbol{x}_s) = \mathcal{N}\big(\alpha_{s,t}\boldsymbol{x}_s,\ \sigma_{s,t}^2\boldsymbol{I}\big). \tag{30}$$

The product of two Gaussians in $\boldsymbol{x}_s$ is proportional to $\exp(-\frac{1}{2}\boldsymbol{x}_s^\top \boldsymbol{\Lambda}\boldsymbol{x}_s + \boldsymbol{\eta}^\top \boldsymbol{x}_s)$ with precision and information vector:

$$\boldsymbol{\Lambda} = \frac{1}{s^2}\boldsymbol{I} + \frac{\alpha_{s,t}^2}{\sigma_{s,t}^2}\boldsymbol{I}, \tag{31}$$

$$\boldsymbol{\eta} = \frac{(1-s)\boldsymbol{x}_0}{s^2} + \frac{\alpha_{s,t}\boldsymbol{x}_t}{\sigma_{s,t}^2}. \tag{32}$$

We first compute $\sigma_{s,t}^2 = t^2 - \alpha_{s,t}^2 s^2$. Defining $D := t^2(1-s)^2 - s^2(1-t)^2$, we note that:

$$D = \big[t(1-s) - s(1-t)\big]\big[t(1-s) + s(1-t)\big] = (t-s)(t+s-2ts), \tag{33}$$

so $\sigma_{s,t}^2 = D/(1-s)^2$.

Precision.

$$\Lambda = \frac{1}{s^2} + \frac{(1-t)^2/(1-s)^2}{D/(1-s)^2} = \frac{1}{s^2} + \frac{(1-t)^2}{D} = \frac{D + s^2(1-t)^2}{s^2 D} = \frac{t^2(1-s)^2}{s^2 D}, \tag{34}$$

where the last step uses $D + s^2(1-t)^2 = t^2(1-s)^2$.

Posterior variance.

$$C^2 = \Lambda^{-1} = \frac{s^2 D}{t^2(1-s)^2} = \frac{s^2(t-s)(t+s-2ts)}{t^2(1-s)^2}. \tag{35}$$

Posterior mean. The information vector is $\boldsymbol{\eta} = (1-s)\boldsymbol{x}_0/s^2 + \alpha_{s,t}\boldsymbol{x}_t/\sigma_{s,t}^2$. Computing $\alpha_{s,t}/\sigma_{s,t}^2 = [(1-t)/(1-s)] \cdot (1-s)^2/D = (1-t)(1-s)/D$, the posterior mean is:

$$\begin{aligned}\boldsymbol{\mu}_{\text{post}} &= C^2 \cdot \boldsymbol{\eta} \\ &= \frac{s^2 D}{t^2(1-s)^2}\left[\frac{(1-s)\boldsymbol{x}_0}{s^2} + \frac{(1-t)(1-s)\boldsymbol{x}_t}{D}\right] \\ &= \frac{D\,\boldsymbol{x}_0}{t^2(1-s)} + \frac{s^2(1-t)\,\boldsymbol{x}_t}{t^2(1-s)} \\ &= A(t,s)\,\boldsymbol{x}_t + B(t,s)\,\boldsymbol{x}_0. \end{aligned} \tag{36}$$

In the finetuning recipe (§ 3.3), $\boldsymbol{x}_0$ is replaced by the predicted clean sample $\hat{\boldsymbol{x}}_0 = \boldsymbol{x}_t - t \cdot \boldsymbol{v}_\theta$, where $\boldsymbol{v}_\theta$ is the pretrained velocity prediction. □

Numerical values for a 4-step schedule. Table 3 provides representative values of A, B, and C for the default 4-step schedule used in our experiments.

Table 3: Posterior coefficients for a representative 4-step schedule.

Step	t	s	$A(t,s)$	$B(t,s)$	$C(t,s)$
1	1.000	0.754	0.140	0.614	0.371
2	0.754	0.509	0.271	0.496	0.362
3	0.509	0.263	0.470	0.362	0.297
4	0.263	0.020	0.948	0.049	0.049

A.6 Trajectory Covariance Matrix

The self-refinement step (Eq. (12)) uses a trajectory covariance matrix $\boldsymbol{S}$ whose (i,j)-th entry is the covariance between $\boldsymbol{x}_{t_i}$ and $\boldsymbol{x}_{t_j}$ conditioned on $\boldsymbol{x}_0$.

Proposition 3. *Under the Markov forward process, for any two timesteps t_i, t_j in the trajectory:*

$$\operatorname{Cov}(\boldsymbol{x}_{t_i}, \boldsymbol{x}_{t_j} \mid \boldsymbol{x}_0) = \frac{\min(t_i,t_j)^2\big(1-\max(t_i,t_j)\big)}{1-\min(t_i,t_j)} \cdot \boldsymbol{I}. \tag{37}$$

Proof. Without loss of generality, assume $t_i \le t_j$ (so $\min = t_i$, $\max = t_j$). Write $\boldsymbol{x}_{t_i} = (1-t_i)\boldsymbol{x}_0 + \boldsymbol{\xi}_i$ where $\boldsymbol{\xi}_i \sim \mathcal{N}(\boldsymbol{0}, t_i^2\boldsymbol{I})$ is the noise component. By the Markov property (Eq. (3)):

$$\boldsymbol{x}_{t_j} = \alpha_{t_i,t_j}\,\boldsymbol{x}_{t_i} + \sigma_{t_i,t_j}\,\boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}) \text{ independent of } \boldsymbol{x}_{t_i}. \tag{38}$$

Therefore:

$$\begin{aligned}\operatorname{Cov}(\boldsymbol{x}_{t_i}, \boldsymbol{x}_{t_j} \mid \boldsymbol{x}_0) &= \operatorname{Cov}(\boldsymbol{\xi}_i,\ \alpha_{t_i,t_j}\boldsymbol{\xi}_i + \sigma_{t_i,t_j}\boldsymbol{\epsilon}) \\ &= \alpha_{t_i,t_j}\operatorname{Var}(\boldsymbol{\xi}_i) \\ &= \frac{1-t_j}{1-t_i} \cdot t_i^2 \cdot \boldsymbol{I} = \frac{t_i^2(1-t_j)}{1-t_i} \cdot \boldsymbol{I}. \end{aligned} \tag{39}$$

Since $t_i = \min(t_i,t_j)$ and $t_j = \max(t_i,t_j)$, this matches Eq. (37). The diagonal case ($i=j$) gives $\operatorname{Var}(\boldsymbol{x}_{t_i} \mid \boldsymbol{x}_0) = t_i^2$, consistent with the formula via $\lim_{t_j \to t_i} t_i^2(1-t_j)/(1-t_i) = t_i^2$. □

Self-refinement update. Using this covariance, the self-refinement (Eq. (12)) applies a covariance-weighted gradient step:

$$\hat{\boldsymbol{x}} \leftarrow \frac{\hat{\boldsymbol{x}} - \boldsymbol{S}\,\nabla_{\hat{\boldsymbol{x}}}\mathcal{L}_{\text{NTM}}}{1-\boldsymbol{t}}, \tag{40}$$

where the matrix-vector product $\boldsymbol{S}\,\nabla_{\hat{\boldsymbol{x}}}\mathcal{L}$ couples the gradient across timesteps according to their noise correlation. Unlike per-step independent correction (which would use only the diagonal $\operatorname{Var}(\boldsymbol{x}_{t_i}) = t_i^2$), this expression propagates correlated corrections across timesteps.

B Algorithm Pseudocode

B.1 Training

Algorithm 1 NTM Training (single iteration)

Require: Clean data $\boldsymbol{x}_0$, condition $\boldsymbol{y}$ (text or class label), number of steps T, noise range $[t_{\min}^{\text{lo}}, t_{\min}^{\text{hi}}]$

Sample per-example minimum noise: $t_{\min} \sim \text{Uniform}[t_{\min}^{\text{lo}}, t_{\min}^{\text{hi}}]$
Compute shifted timestep schedule: $(t_0, t_1, \ldots, t_T)$ with $t_0 = t_{\min}$
Sample $\boldsymbol{\epsilon}_{\text{init}} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$; set $\boldsymbol{x}_{t_0} = (1 - t_0)\boldsymbol{x}_0 + t_0\boldsymbol{\epsilon}_{\text{init}}$
Forward trajectory: for $k = 0, \ldots, T-1$:
    $\boldsymbol{\epsilon}_k \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
    $\boldsymbol{x}_{t_{k+1}} = \alpha_{t_k, t_{k+1}}\boldsymbol{x}_{t_k} + \sigma_{t_k, t_{k+1}}\boldsymbol{\epsilon}_k$
Transporter: $\boldsymbol{u}_{t_k} = f_{\mathcal{T}}(\boldsymbol{x}_{t_k}, t_k)$ for $k < T$; $\boldsymbol{u}_{t_T} = \boldsymbol{x}_{t_T}$; accumulate log-determinants
Predictor (trajectory coupling): for $k = 1, \ldots, T$:
    $(\boldsymbol{\mu}_{\mathcal{P}}^{(k)}, \boldsymbol{\sigma}_{\mathcal{P}}^{(k)}) = \text{DeepBlock}(\boldsymbol{u}_{t_k}, t_k, t_{k-1}, \boldsymbol{y})$
    $\boldsymbol{z}_k = (\boldsymbol{u}_{t_{k-1}} - \boldsymbol{\mu}_{\mathcal{P}}^{(k)})/\boldsymbol{\sigma}_{\mathcal{P}}^{(k)}$
NTM loss: $\mathcal{L}_{\text{NTM}} = \sum_{k=1}^{T} \left[\frac{1}{2}\|\boldsymbol{z}_k\|^2 + \sum_n \log \boldsymbol{\sigma}_{\mathcal{P}}^{(k,n)}\right] - \sum_\ell \log|\det J_{f_{\mathcal{T}}^{(\ell)}}|$
FM auxiliary loss: $\mathcal{L}_{\text{aux}} = \sum_k \|\boldsymbol{\mu}_{\mathcal{P}}^{(k)} - \boldsymbol{\mu}_{\text{FM}}^{(k)}\|^2$ if finetuning; otherwise 0
Update θ via $\nabla_\theta(\mathcal{L}_{\text{NTM}} + \lambda\mathcal{L}_{\text{aux}})$; keep the teacher frozen

B.2 Sampling

Algorithm 2 NTM Sampling

Require: Condition $\boldsymbol{y}$, transitions T, guidance scale w, schedule $0 < t_0 < \cdots < t_T = 1$, sampling route

Sample initial noise: $\hat{\boldsymbol{u}}_{t_T} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
Predictor reverse (parallel over spatial, sequential over k):
for $k = T, T-1, \ldots, 1$ **do**
    $\boldsymbol{z}_k \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
    $(\boldsymbol{\mu}_{\mathcal{P}}^{(k)}, \boldsymbol{\sigma}_{\mathcal{P}}^{(k)}) = \text{DeepBlock}(\hat{\boldsymbol{u}}_{t_k}, t_k, t_{k-1}, \boldsymbol{y})$
    (*If CFG*) Apply guidance: $(\boldsymbol{\mu}_{\mathcal{P}}^{(k)}, \boldsymbol{\sigma}_{\mathcal{P}}^{(k)}) \leftarrow \text{CFG}(\cdot, w)$
    $\hat{\boldsymbol{u}}_{t_{k-1}} = \boldsymbol{z}_k \cdot \boldsymbol{\sigma}_{\mathcal{P}}^{(k)} + \boldsymbol{\mu}_{\mathcal{P}}^{(k)}$
end for
if using the learned-denoiser route **then**
    $\hat{\boldsymbol{x}}_0^{\text{out}} = g_\phi(\hat{\boldsymbol{u}}_{t_0}, \boldsymbol{y})$ {one forward pass; bypass inverse and refinement}
else
    $\hat{\boldsymbol{x}}_{t_k} = f_{\mathcal{T}}^{-1}(\hat{\boldsymbol{u}}_{t_k}, t_k)$ for $k < T$; $\hat{\boldsymbol{x}}_{t_T} = \hat{\boldsymbol{u}}_{t_T}$
    $\hat{\boldsymbol{x}}^{\text{den}} \leftarrow$ Algorithm 3 applied to this generated trajectory
    $\hat{\boldsymbol{x}}_0^{\text{out}} = \hat{\boldsymbol{x}}_{t_0}^{\text{den}}$
end if
Decode: $\text{image} = \text{Decoder}(\hat{\boldsymbol{x}}_0^{\text{out}})$ {FAE or VAE, according to the setting}
return image

B.3 Trajectory Self-Refinement

Algorithm 3 Trajectory Self-Refinement

Require: Generated trajectory $\hat{\boldsymbol{x}} = (\hat{\boldsymbol{x}}_{t_0}, \ldots, \hat{\boldsymbol{x}}_{t_T})$, frozen NTM model, schedule $(t_0, \ldots, t_T)$

1: Enable gradients w.r.t. $\hat{\boldsymbol{x}}$
2: Forward pass through NTM: compute the negative joint log-density, including the fixed terminal prior
3: Compute gradient: $\boldsymbol{g} = -\nabla_{\hat{\boldsymbol{x}}} \log p_\theta(\hat{\boldsymbol{x}})$
4: (*Optional*) Percentile-based gradient clipping on $\boldsymbol{g}$
5: Compute covariance for $i, j < T$: $[\boldsymbol{S}]_{ij} = \min(t_i, t_j)^2(1 - \max(t_i, t_j))/(1 - \min(t_i, t_j))$
6: Set the terminal row and column of $\boldsymbol{S}$ to zero; keep the pure-noise endpoint fixed
7: Covariance-weighted correction: $\hat{\boldsymbol{x}} \leftarrow \hat{\boldsymbol{x}} - \boldsymbol{S}\,\boldsymbol{g}$ {couples gradients across timesteps}
8: Normalize only nonterminal states: $\hat{\boldsymbol{x}}^{\text{den}}_{t_k} = \hat{\boldsymbol{x}}_{t_k}/(1 - t_k)$ for $k < T$
9: **return** $(\hat{\boldsymbol{x}}^{\text{den}}_{t_0}, \ldots, \hat{\boldsymbol{x}}^{\text{den}}_{t_{T-1}})$

C Implementation Details

C.1 Model Architecture

Table 4 summarizes the architectural specifications of the NTM models used in our experiments.

Table 4: Architectural specifications of NTM models.

	From scratch	**Finetuned**
Hidden dimension	3072	3072
Number of blocks	3	3
Layers per block	[4, 4, 24]	[4, 4, 24]
Transporter	Blocks 1–2 (4 layers each)	Blocks 1–2 (4 layers each)
Predictor	Block 3 (24 layers)	FLUX.2-klein (4B)
Patch size	1	2
KV heads	8	8
Positional encoding	2D RoPE	2D RoPE
Transporter scan order	Alternating (L→R, R→L)	Alternating (L→R, R→L)
Pretrained backbone	—	FLUX.2-klein (4B)
Denoising mode	—	`true_reverse`
`no_delta_mean`	—	1 ($\boldsymbol{\mu}_{\mathcal{P}} = \boldsymbol{\mu}_{\text{post}}$)

Predictor. In the from-scratch setting, the predictor uses non-causal attention over spatial positions. For each consecutive pair, it takes the noisier representation $\boldsymbol{u}_{t_k}$ and text embeddings $\boldsymbol{y}$, and predicts the coupling parameters for $\boldsymbol{u}_{t_{k-1}}$. Training pairs can be evaluated in parallel without conditioning a reverse kernel on its clean-side target. Timestep conditioning is provided through additive sinusoidal embeddings.

In the finetuned setting, the predictor wraps a pretrained flow-matching backbone (FLUX.2). The backbone remains trainable. Its velocity prediction defines the posterior mean in Eq. (13), and its last hidden states feed a zero-initialized scale projection. The general residual parameterization can predict $(\delta_\mu, \delta_\sigma)$, but the reported configuration uses `no_delta_mean=1` and therefore disables the separate δ_μ correction; only δ_σ is used. At initialization the scale correction is zero and the student reproduces the pretrained Gaussian posterior. Disabling the residual mean head does not freeze the backbone or its predicted mean.

Transporter. The transporter has two TarFlow-style causal autoregressive flow blocks, each with four Transformer layers. Blocks alternate between identity and flip permutations (left-to-right and right-to-left scan directions) for better spatial mixing. At the highest noise level ($t \approx 1$), the transporter is skipped (identity transform) since the input is nearly isotropic Gaussian and the spatial AR coupling would be uninformative.

C.2 Training Hyperparameters

The hyperparameters are listed in Table 5.

Table 5: Training hyperparameters.

	From Scratch (ImageNet)	**Finetuned**
Optimizer	AdamW	AdamW
(β_1, β_2)	$(0.9, 0.95)$	$(0.9, 0.95)$
Weight decay	10^{-4}	10^{-4}
Peak learning rate	10^{-4}	5×10^{-5}
Minimum learning rate	10^{-6}	10^{-6}
LR schedule	Cosine with warmup	Cosine with warmup
Precision	bfloat16	bfloat16
Distributed strategy	FSDP2	FSDP2
Default transitions (T)	4	4
$t_{\min}$ range	$\mathrm{Uniform}[0.0, 0.05]$	$\mathrm{Uniform}[0.0, 0.05]$
CFG dropout	10%	10%
FM aux loss weight (λ)	—	2.5
FM aux loss type	—	MSE
λ schedule	—	Fixed

C.3 Denoiser Architecture

The learned denoiser g_ϕ (§ 3.4) is a lightweight Transformer that takes the predictor output $\boldsymbol{u}_{t_0}$ at the cleanest level as input and produces a denoised image $\hat{\boldsymbol{x}}_0^{\mathrm{den}}$ in a single forward pass. The network approximates the explicit refinement target from this endpoint representation using MSE. It is not constrained to be invertible, and its output density is not evaluated.

- **Position encoding**: 2D rotary embeddings over spatial (row, column) dimensions.
- **Attention**: Full non-causal attention over all spatial positions.
- **Conditioning**: Text embeddings $\boldsymbol{y}$ are concatenated to the input sequence.
- **Output**: A single predicted clean image in patch space.
- **Training**: After the main NTM model converges, the frozen model generates targets via trajectory score denoising, and g_ϕ is trained with MSE loss (Eq. (16)).

C.4 Timestep Schedule

We use a shifted timestep schedule [17] that adapts to the input sequence length. Given T denoising steps, the base schedule is:

$$\tilde{\sigma}_k = \frac{k}{T}, \quad k = 1, \ldots, T, \tag{41}$$

which is then shifted via a sequence-length-dependent parameter μ:

$$\sigma_k = \frac{e^\mu}{e^\mu + 1/\tilde{\sigma}_k - 1}, \quad \mu = 0.5 + 0.65 \cdot \frac{L_{\mathrm{seq}} - 256}{4096 - 256}, \tag{42}$$

where L_{seq} is the spatial sequence length (number of patches). The final schedule is $(t_{\min}, \sigma_1, \ldots, \sigma_T)$ in ascending order, with $t_{\min}$ drawn per-sample from $\mathrm{Uniform}[t_{\min}^{\mathrm{lo}}, t_{\min}^{\mathrm{hi}}]$ for robustness across noise levels during training.

C.5 Classifier-Free Guidance

At inference, we use a logits-guided formulation of classifier-free guidance [27] that operates on the coupling parameters rather than the predicted sample. Given conditional predictions $(\boldsymbol{\mu}_c, \boldsymbol{\sigma}_c)$ and

unconditional predictions $(\boldsymbol{\mu}_u, \boldsymbol{\sigma}_u)$ from the predictor, the guided parameters are:

$$s = \left(\frac{\boldsymbol{\sigma}_c}{\boldsymbol{\sigma}_u}\right)^2 \quad \text{(clipped to } [0,1]\text{)}, \tag{43}$$

$$\boldsymbol{\sigma}_{\text{eff}} = \frac{\boldsymbol{\sigma}_c}{\sqrt{1 + w - w \cdot s}}, \tag{44}$$

$$\boldsymbol{\mu}_{\text{eff}} = \frac{(1+w)\,\boldsymbol{\mu}_c - w \cdot s \cdot \boldsymbol{\mu}_u}{1 + w - w \cdot s}, \tag{45}$$

where w is the guidance scale. This formulation is inspired by the logit-space interpretation: it corresponds to $(1+w)\log p_c - w \log p_u$ applied to the Gaussian coupling in u-space, which naturally adjusts both the mean and scale (unlike standard linear guidance that only modifies the mean).

D Evaluation Benchmarks

D.1 GenEval

GenEval [20] is a compositional text-to-image evaluation benchmark that tests fine-grained generation capabilities across six task categories:

- **Single object**: generating a single named object correctly.
- **Two objects**: generating two distinct objects in the same scene.
- **Counting**: producing the exact number of objects specified.
- **Colors**: assigning the correct color to objects.
- **Position**: placing objects in the specified spatial relationship (e.g., "left of", "above").
- **Color attribution**: binding the correct color to the correct object when multiple colored objects are described.

Each task is scored by an object-detection model that verifies whether the specified objects, attributes, and relations are present. The overall score is the average accuracy across all six tasks. We generate 4 images per prompt and report the average detection rate.

D.2 DPG-Bench

DPG-Bench [30] (Dense Prompt Graph Benchmark) evaluates text-to-image alignment using long, detailed prompts that describe complex scenes with multiple entities, attributes, and relations. Unlike GenEval which uses short compositional prompts, DPG-Bench tests whether models can faithfully follow dense, paragraph-length descriptions. The benchmark uses visual question answering to assess semantic elements of each prompt; its official evaluator uses mPLUG. The benchmark reports scores across five L1 categories:

- **Attribute** (color, shape, size, texture, other)
- **Entity** (part, state, whole)
- **Global** (overall scene coherence)
- **Other** (counting, text rendering)
- **Relation** (spatial, non-spatial)

The overall score is aggregated using the benchmark's question-dependency scoring protocol and reported as a percentage.

D.3 Class-Conditional ImageNet

As a proof-of-concept for training NTM from scratch, we evaluate on class-conditional ImageNet 256×256 using the FAE latent space [18] (16× spatial compression, 32-dim latents). Table 6 reports FID-50K for NTM at different step counts alongside representative baselines.

Table 6: **Class-conditional ImageNet 256×256** (FID-50K). Counts denote reverse transitions for NTM, sampling steps for diffusion models, and spatial positions or scales for autoregressive models. The generation structures differ and these counts are not matched latency measurements. The base NTM uses NLL training without distribution-level losses.

Method	Type	#Params	Steps	FID↓
DiT-XL/2 [40]	DM	675M	250	2.27
SiT-XL [37]	DM	675M	250	2.06
LlamaGen [49]	AR	3.1B	256	2.18
VAR [50]	AR	2.0B	10	1.73
DART [23]	AR	820M	16	3.82
TarFlow [56]	NF	1.4B	1024	5.56
STARFlow (VAE) [25]	NF	1.4B	1024	2.40
STARFlow (FAE) [18]	NF	1.4B	256	2.67
NTM (Ours)	NF	1.4B	4	3.83
NTM (Ours)	NF	1.4B	8	3.24
NTM (Ours)	NF	1.4B	16	2.80

NTM achieves 2.80 FID with 16 reverse transitions, close to STARFlow (FAE) at 2.67 with 256 spatial autoregressive positions. These annotations describe different generation structures and are not matched latency measurements. The base trajectory model is trained with the exact transition NLL objective, without distribution-level losses such as adversarial or perceptual objectives. This describes base-model training; the separate learned denoiser, when used, is trained with MSE as specified in § 3.4. Recent work [55; 54; 53] has shown that distribution-level finetuning can substantially boost few-step generators beyond their base performance; NTM's stable exact-likelihood training makes it a natural candidate for such post-training enhancement, which we leave for future work.

E Additional Qualitative Results

We show additional samples from our trained NTM models in Figure 8.

F Broader Impact

NTM combines high-quality few-transition sampling with a base model whose transition and joint-trajectory densities can be evaluated exactly. The fast learned-denoiser output is not itself assigned an evaluated exact density. While improved generative models have many beneficial applications—including creative tools, data augmentation, and scientific visualization—they also raise concerns around potential misuse for generating misleading or harmful content. Tractable base-model densities may support future research on model auditing. Applications such as anomaly detection and content verification are not evaluated here, and exact base-model density alone does not establish the authenticity or safety of generated content. We encourage the research community to develop complementary safeguards, including watermarking and provenance tracking, alongside advances in generative modeling.

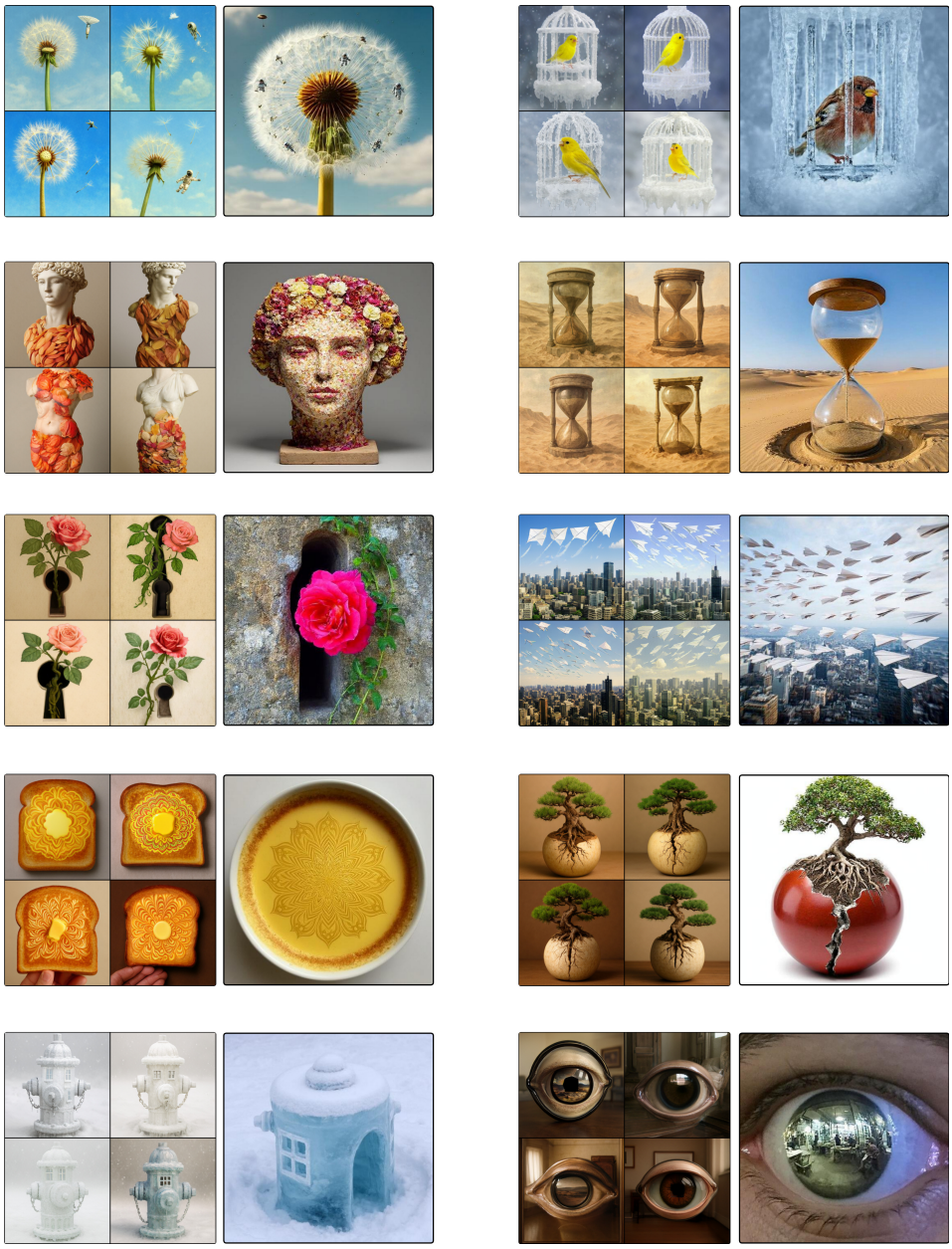


Figure 8: Additional examples from NTM trained from scratch (left) and fine-tuned from flow matching (right) under the same text prompts.

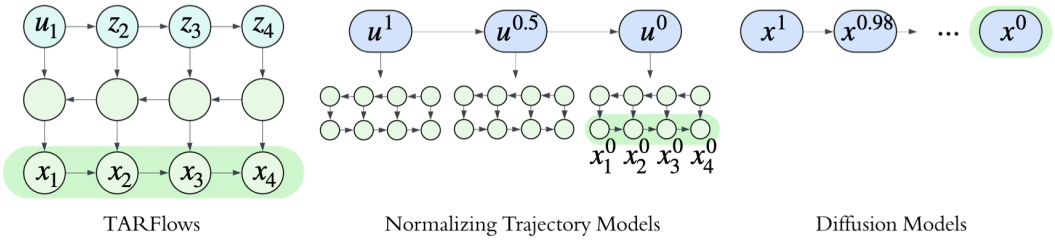


Figure 9: **Comparison between TARFlows, Normalizing Trajectory Models, and Diffusion Models.** Subscripts denote spatial patch indices, and superscripts denote trajectory timesteps, with $\boldsymbol{x}^1, \boldsymbol{u}^1$ representing pure Gaussian noise and $\boldsymbol{x}^0, \boldsymbol{u}^0$ the clean data.

G Discussion

NTM as an interpolation between normalizing flows and flow matching. STARFlow [25] directly models the marginal image distribution $p(\boldsymbol{x})$ by decomposing it via a deep spatial autoregressive flow within a single generation step—generation traverses its spatial autoregressive positions (256 for the cited model) through deep and shallow blocks. At the other extreme, flow matching models a velocity field whose ODE integration requires many numerical integration steps for high quality. NTM occupies a middle ground: it explicitly models each intermediate conditional $p(\boldsymbol{x}_s \mid \boldsymbol{x}_t)$ along a T-step denoising trajectory as a normalizing flow, as shown in Fig. 9.



Figure 10: **Failure Case:** NTM with $T{=}1$ produces degraded outputs in the tested configuration. Prompt: *a corgi dog.*

The key architectural tradeoff is where to place depth. STARFlow concentrates all capacity *within* a single step via deep spatial AR blocks; NTM distributes capacity *across* multiple denoising steps, using a shallow transporter (2 blocks $\times$ 4 layers) at each step paired with a deep trajectory-level predictor. This trades per-step expressiveness for multi-step structure: the predictor reasons across timestep levels while each transporter handles only the local non-Gaussian residual within one step. As a result, the per-step normalizing flow in NTM can be lightweight because each step only needs to capture the conditional $p(\boldsymbol{x}_s \mid \boldsymbol{x}_t)$—which is simpler than the full marginal $p(\boldsymbol{x})$—while the deep predictor captures the bulk of the denoising signal in u-space.

Why single-step generation remains challenging. As shown in Fig. 10, NTM with $T{=}1$ produces severely degraded outputs. This behavior is consistent with limited transporter capacity, although the present experiments do not isolate capacity from optimization effects. At $T{=}1$, the entire non-Gaussian structure of the data distribution must be captured by the shallow transporter alone—the predictor reduces to a single-step Gaussian coupling. This configuration is effectively a STARFlow-like architecture with a parallel (non-causal) prior, but with far fewer transporter layers than STARFlow's deep blocks (8 layers vs. STARFlow's 24+ layers per block $\times$ multiple blocks). Deeper transporters may improve single-step expressiveness, but would also increase the cost of autoregressive decoding; evaluating this tradeoff remains future work.

For finetuning, the $T{=}1$ setting introduces additional challenges: the mean-alignment auxiliary loss (Eq. (15)) was designed to anchor the predictor to a multi-step denoising trajectory, and collapsing to a single step fundamentally changes the training dynamics.

Implications. NTM's sweet spot is $T{=}4$–8: enough steps for the shallow transporter to distribute the non-Gaussian modeling across the trajectory, while the deep predictor handles cross-timestep reasoning efficiently in parallel. The architecture naturally admits a spectrum—deeper transporters with fewer steps, or shallower transporters with more steps—offering a principled way to trade off sequential computation for generation quality. Pushing toward single-step generation with exact likelihood remains an open challenge that may require fundamentally different architectural choices, such as adaptive-depth transporters or progressive capacity allocation across the trajectory.

G.1 Likelihood Scope and Inference Accounting

The normalized base model evaluates exact reverse-transition and joint-trajectory densities for a fixed schedule. The endpoint marginal requires integrating out intermediate states; the fixed forward process yields an ELBO for this marginal. The learned denoiser is trained with MSE, is not constrained to be invertible, and does not provide an evaluated output density. All NTM entries in the main text-to-image comparison use four sequential predictor transitions followed by one learned-denoiser pass. This forward-only route bypasses autoregressive transporter inversion and test-time input gradients. The throughput measurements in Table 2 use a B200 GPU at batch size one, giving a 9.4-fold speedup; LPIPS is measured against the explicit refinement output.

G.2 Interpreting Finetuning

The finetuned model scores lower on GenEval (0.76 versus 0.82), but higher on DPG-Bench (83.38 versus 79.64). The settings use different latent representations and resolutions, so these results do not isolate a causal effect of initialization. Mean alignment is a soft training-time penalty: the teacher is frozen, while the student backbone, scale correction, and transporter remain trainable. Disabling a separate residual mean head does not freeze the backbone.

The from-scratch model uses FAE latents at 256×256, while finetuning preserves the native FLUX VAE at $512{\times}512$. The latter preserves the representation of the pretrained backbone during conversion. GenEval differences are concentrated in position and attribute binding, whereas the finetuned model is stronger on dense-prompt DPG-Bench evaluation. This suggests different compositional profiles rather than a uniform quality regression; the experiments do not isolate their cause.

G.3 Evaluation Scope and Remaining Limitations

The likelihood contribution is algorithmic: transition and joint-trajectory densities are tractable and provide the refinement score. We do not report a held-out density-ranking result. A trajectory NLL depends on the chosen schedule and is not directly comparable to STARFlow's marginal image-latent NLL. Both models also use lossy latent representations, so these quantities are not pixel-space bits per dimension.

For a fixed schedule, write $\boldsymbol{x}_{0:T}$ for its states (indices here label schedule positions) and $q(\boldsymbol{x}_{1:T} \mid \boldsymbol{x}_0)$ for the fixed forward process. Then

$$\log p_\theta(\boldsymbol{x}_0) \geq \mathbb{E}_{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)} \left[\log p_\theta(\boldsymbol{x}_{0:T}) - \log q(\boldsymbol{x}_{1:T} \mid \boldsymbol{x}_0)\right]. \tag{46}$$

Since q is fixed, minimizing expected joint NLL maximizes this bound up to a parameter-independent term. At a positive minimum noise level, $\boldsymbol{x}_0$ in this expression is the noisy latent endpoint; the subsequent denoised output is a separate readout.

The quantitative trajectory-length study covers $T \in \{4, 8, 16\}$; $T = 4$ is the lowest tested value reaching the reported high-quality regime. We do not report quantitative one- or two-transition quality results. The present comparison emphasizes generation from normalized trajectory models. DMD2, LCM, Hyper-SD, and distilled FLUX.2-klein provide complementary implicit-distillation comparisons; matched quality–latency evaluation against these methods remains future work. The measured 0.20 versus 1.88 img/s comparison concerns two routes of the same NTM, not those external baselines. Peak inference memory was not measured in this comparison.

The text-to-image results rely on an internal licensed corpus of approximately 70M pairs, including CC12M, which limits independent reproduction. The ImageNet experiment provides a separate public-data test of the core formulation. The evaluated architecture uses two transporter blocks; a controlled block-count ablation is not reported. Additional transporter depth could increase conditional expressiveness but would increase explicit autoregressive inversion cost.

G.4 Connection to Stochastic Flow Maps

As an additional interpretation, NTM can be viewed as a stochastic flow map implemented with conditional normalizing flows. Its transition sampler can be written as

$$F^{\theta}_{t\to s}(\boldsymbol{x}_t, \boldsymbol{z}; \boldsymbol{y}) = f_{\mathcal{T}}^{-1}(\boldsymbol{\mu}_{\mathcal{P}}(f_{\mathcal{T}}(\boldsymbol{x}_t, t), t, s, \boldsymbol{y}) + \boldsymbol{\sigma}_{\mathcal{P}}(f_{\mathcal{T}}(\boldsymbol{x}_t, t), t, s, \boldsymbol{y}) \odot \boldsymbol{z},\ s)\,. \tag{47}$$

For a fixed noisy state and condition, this maps fresh Gaussian noise to a sample of the learned coarse reverse conditional. Invertibility in the noise variable makes that conditional density tractable. This viewpoint connects NTM to stochastic flow-map models without changing its formulation or training objective.

This interpretation concerns transition laws, as in weak stochastic flow-map models [29; 33]. It does not identify the learned maps with a strong solution driven by a shared Brownian path [38]. At the population optimum, matching the reverse conditionals of a common Markov process recovers its finite-schedule law. Our finite-parameter training objective, however, does not enforce Chapman–Kolmogorov consistency across arbitrary schedules: a learned direct $t \to s$ kernel need not equal the composition through a separately learned intermediate time. Exact normalization for each chosen schedule does not imply that stronger consistency property. Enforcing it is a possible future direction.

Shared coordinates and composition. Write $f_t(\boldsymbol{x}) = f_{\mathcal{T}}(\boldsymbol{x}, t)$ and let $H^{\theta}_{t\to s}(\boldsymbol{u}, \boldsymbol{z}) = \boldsymbol{\mu}_{\mathcal{P}}(\boldsymbol{u}, t, s, \boldsymbol{y}) + \boldsymbol{\sigma}_{\mathcal{P}}(\boldsymbol{u}, t, s, \boldsymbol{y}) \odot \boldsymbol{z}$, with the condition $\boldsymbol{y}$ suppressed below. Then the stochastic map is $F^{\theta}_{t\to s} = f_s^{-1} \circ H^{\theta}_{t\to s} \circ f_t$. For $s < r < t$ on a fixed schedule and independent innovations $\boldsymbol{z}_1, \boldsymbol{z}_2$, its composition satisfies

$$\begin{aligned} &F^{\theta}_{r\to s}\big(F^{\theta}_{t\to r}(\boldsymbol{x}_t, \boldsymbol{z}_1), \boldsymbol{z}_2\big) \\ &\qquad = f_s^{-1}\big(H^{\theta}_{r\to s}\big(H^{\theta}_{t\to r}(f_t(\boldsymbol{x}_t), \boldsymbol{z}_1), \boldsymbol{z}_2\big)\big)\,. \end{aligned} \tag{48}$$

The intermediate $f_r \circ f_r^{-1}$ cancels exactly. Here each occurrence of f_r uses the same timestep and schedule conditioning; the cancellation is within a chosen schedule. This algebra explains why sampling a sequence of normalized x-space transitions does not require materializing every intermediate x-space state. One can sample the same composed base-model endpoint by running the stochastic predictor recurrence in u-space and applying only the final inverse. It does not establish equality with a separately learned direct $t \to s$ transition, which is the distinct consistency question discussed above.

From a tractable stochastic map to a fast readout. A generic conditional normalizing flow can still be slow to invert, so the stochastic flow-map interpretation by itself is not a speed guarantee. NTM allocates most capacity to the spatially parallel predictor and composes transitions in the shared coordinates. The remaining endpoint inverse is spatially autoregressive. Explicit trajectory refinement additionally reconstructs trajectory states and differentiates their joint density, exploiting correlations across time. The learned denoiser approximates this corrected endpoint from the final predictor state, bypassing both autoregressive inversion and test-time input gradients. The measured four-transition-plus-readout route therefore combines finite-interval stochastic transport with amortized correction; the reported $9.4\times$ gain compares this route to explicit inversion and refinement of the same model.

Expressive stochasticity and conditional uncertainty. A coarse reverse kernel must account for multiple clean states consistent with the same noisy observation. A deterministic map chooses one output for a fixed input, and an affine Gaussian stochastic map restricts the conditional family even though it can generate diverse samples. In NTM, the inverse transporter nonlinearly reshapes the Gaussian innovation into an expressive reverse conditional while change of variables retains its density. This does not guarantee that the learned model captures all conditional modes; it specifies a richer normalized family and an objective for fitting it. In contrast to an implicit stochastic map, the explicit density also supplies the joint trajectory score used to supervise the readout. That score-based training signal is an additional consequence of the NF parameterization, beyond the ability to draw a stochastic sample.

Relation to other flow-map constructions. Neural Stochastic Flows [33] establish conditional normalizing flows as a parameterization of stochastic transition laws. Diamond Maps [29] use stochastic maps for efficient reward alignment. NTM specializes coarse reverse kernels to image generation and uses their joint score as a refinement target. Concurrent NF^{2}M [21] proposes clean molecular configurations with conditional flows and corrects denoising transitions by importance weighting. Concurrent Normalizing Flow Maps [44] learn invertible two-time pushforward maps with endpoint densities and consistency objectives; NTM instead injects fresh per-transition noise and evaluates a joint trajectory density. The non-invertible denoiser output is not assigned an exact likelihood.

H Extended Related Work

Normalizing flows for image generation. Normalizing flows [15; 16; 42; 31] learn invertible mappings with exact log-likelihood via the change-of-variables formula. Classical approaches struggled to scale to high-resolution images due to the full-dimensional invertibility constraint. TarFlow [56] addressed this by parameterizing autoregressive coupling layers with causal Transformers, enabling flows to leverage modern sequence-modeling architectures. STARFlow [25; 24] further introduced a deep-shallow design—a single deep autoregressive block followed by lightweight shallow blocks—scaling normalizing flows to competitive text-to-image generation. While these methods model the marginal $p(\boldsymbol{x})$ directly, NTM applies normalizing flows to the *conditional* distribution $p(\boldsymbol{x}_s \mid \boldsymbol{x}_t)$ at each denoising step. Since conditioning on $\boldsymbol{x}_t$ already constrains the space of plausible images, the per-step flow is simpler than the full marginal and requires fewer blocks.

Non-Gaussian reverse processes. The Gaussian assumption in diffusion reverse steps has been challenged by several works. DDGAN [52] trains a GAN discriminator at each denoising step, enabling larger step sizes by modeling an implicit non-Gaussian conditional. However, GAN-based approaches provide no tractable density, suffer from mode-seeking behavior, and are difficult to scale. Diffusion Normalizing Flow [57] combines normalizing flows with diffusion via neural SDEs, but models the entire generation trajectory as a single continuous flow rather than learning an expressive per-step reverse conditional. Concurrent work [12] also explores normalizing flows with iterative denoising using a different architectural design. NTM models the non-Gaussian reverse via normalizing flows with exact log-likelihood, providing mode-covering training, stable optimization, and a tractable score.

Few-step generation and distillation. Reducing sampling steps is a major research direction. Progressive distillation [45] trains a student to match multi-step teacher outputs in fewer steps. Consistency models [48] learn to map any point on the trajectory directly to the clean image. Distribution matching distillation (DMD) [54] and latent consistency models [36] further improve few-step quality via distributional matching objectives. NFM [6] distills pretrained normalizing flow couplings to train faster flow-matching students. NTM is complementary to distillation approaches with a learned non-Gaussian reverse and trains a denoiser for fast inference.

Score-based denoising and refinement. The connection between denoising and score functions [47] has been exploited for test-time sample improvement. TarFlow [56] introduced adding a small amount of noise and applying the gradient of the NF log-likelihood as a score-based denoiser; STARFlow [25] extended this to latent-space generation. These methods perform independent per-sample denoising. NTM generalizes this to *trajectory-level* denoising: since the generated trajectory is a correlated sequence from the Markov forward process, the NTM loss provides a joint score over all timesteps, and the covariance-weighted gradient correction exploits cross-timestep correlations for more effective refinement than per-sample approaches.

Trajectory-level modeling and flow maps. Several methods model generation across multiple trajectory points rather than per-step. Consistency models [48] and latent consistency models [36] learn to project any noisy point directly to the clean endpoint. FlowMaps [8] generalizes this by learning direct mappings between arbitrary pairs of time points on the probability flow ODE. Mean flows [19] learn one-step generators via flow matching with mean prediction. These methods learn deterministic mappings via regression objectives. NTM is distinct in two ways: it retains a *distributional* model of $p(\boldsymbol{x}_s \mid \boldsymbol{x}_t)$ (not a point estimate), enabling sampling diversity and likelihood

evaluation; and it models the conditional at each step as a normalizing flow rather than collapsing all steps into a single mapping.

Alternative few-step formulations. UDPM [1] jointly denoises and upsamples through dimension-changing transitions. DiffRatio [13] learns a density-ratio signal to train a one-step generator without teacher-score supervision. Transition Matching [46] learns expressive discrete-time stochastic kernels under flexible supervision processes. These approaches offer complementary routes to efficient generation. NTM uses same-dimensional invertible transporters to normalize each coarse reverse transition, then derives a joint trajectory score for an amortized readout.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction clearly state our three contributions: (1) modeling the reverse conditional as a normalizing flow, (2) stable finetuning from pretrained flow-matching models, and (3) a learned denoiser for fast inference. All claims are supported by experimental results in § 4.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discuss limitations in § G, including the sequential nature of transporter inversion, the gap between NLL-only training and distribution-level objectives on ImageNet, and the computational overhead of trajectory score denoising.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: All theoretical results (forward transition marginal preservation, reverse posterior coefficients, trajectory covariance structure) are formally stated as propositions with complete proofs in § A.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: We provide complete architectural specifications (§ C.1), training hyperparameters (§ C.2), timestep schedule details (§ C.4), and CFG formulation (§ C.5). Algorithm pseudocode is provided in § B.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: The text-to-image corpus is internal and is not publicly released. We provide architectural details, hyperparameters, and pseudocode in the appendix; the class-conditional ImageNet experiments use a public benchmark.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Training details are provided in § 4.1 and § C.2, including optimizer, learning rate, batch size, number of GPUs, precision, and all model-specific settings.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: We do not report error bars due to the high computational cost of training large-scale generative models multiple times. Our evaluation metrics (FID-50K, GenEval, DPG-Bench) are computed over large sample sets (1000–50000 images), providing stable estimates.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We report the use of 64 H100 GPUs with FSDP in § 4.1, and provide batch sizes and training duration details.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: This research conforms with the NeurIPS Code of Ethics. We use public ImageNet data and pretrained models in accordance with their licenses, and an internal licensed text-image corpus for text-to-image training.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes]

Justification: We discuss broader impacts in § F, including both beneficial applications and potential misuse concerns, and possible future uses of tractable base-model densities.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: We do not release model weights or datasets at this time. Code release will follow internal review for responsible disclosure.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: We cite all datasets (ImageNet, CC12M), pretrained models (FLUX.2-klein, Qwen-2.5-VL, FAE), and evaluation tools (GenEval, DPG-Bench) used in our experiments, with URLs where applicable.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: This paper does not release new datasets or model assets at submission time.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: This paper does not involve crowdsourcing or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

Justification: This paper does not involve research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: LLMs are not used as a component of the core methodology. The text encoder (Qwen-2.5-VL) is used as a frozen feature extractor for text conditioning, which is standard practice in text-to-image generation.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.