Generating images by diffusion

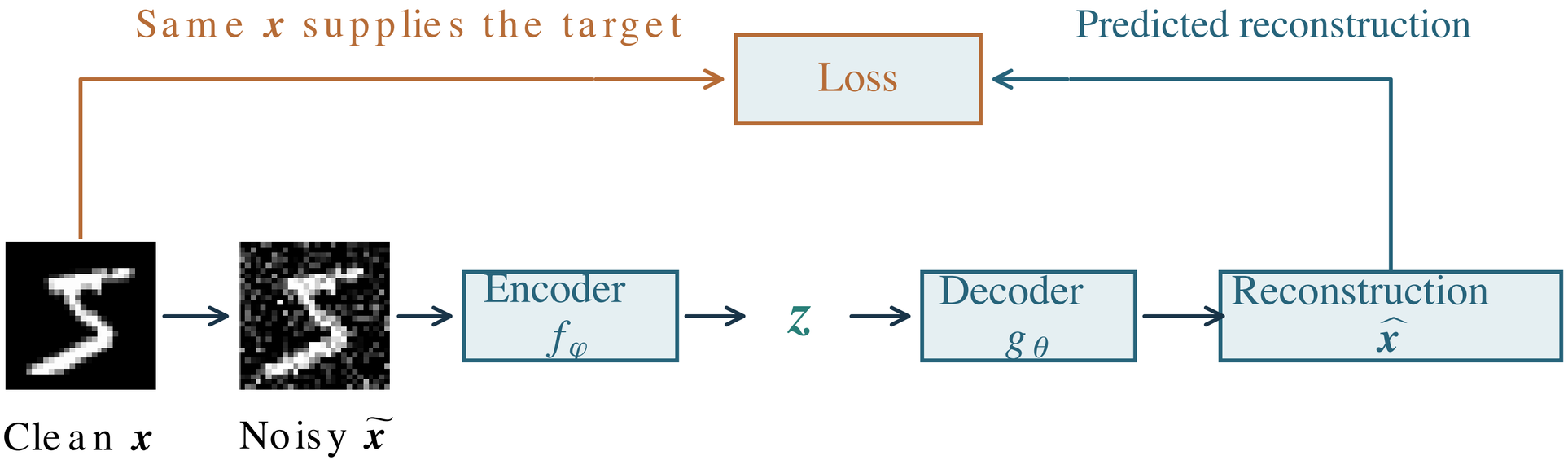
David I. Inouye

Dynamic Content · Last Updated: November 11, 2026

Arc 7 · Image generation and evaluation

Noisy scores → learned fields → multiple levels → SDE/ODE sampling

A denoising autoencoder predicts the clean input



MNIST training image 0; reconstruction is a symbol, not a trained output.

$$\tilde{\boldsymbol{x}}_\sigma = \boldsymbol{x} + \sigma\boldsymbol{\epsilon}, \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, I), \qquad \hat{\boldsymbol{x}} = g_\theta(f_\phi(\tilde{\boldsymbol{x}}_\sigma)).$$

Input: corrupted image. **Target:** original clean image.

Denoising needs a predictor, not a required bottleneck

$$\widehat{\boldsymbol{x}} = D_\theta(\widetilde{\boldsymbol{x}}_\sigma), \qquad \min_\theta \ \mathbb{E}\left[\|\boldsymbol{x} - D_\theta(\boldsymbol{x} + \sigma\boldsymbol{\epsilon})\|^2\right].$$

- Directly map a noisy image to a clean-image prediction of the **same shape**.
- We can use any suitable neural network; an explicit latent code is optional.
- An encoder–decoder or U-Net can still be a useful architecture.

Predicting the clean image and predicting its noise are equivalent

For $\widetilde{\boldsymbol{x}}_\sigma = \boldsymbol{x} + \sigma\boldsymbol{\epsilon}$ at a **fixed** $\sigma > 0$:

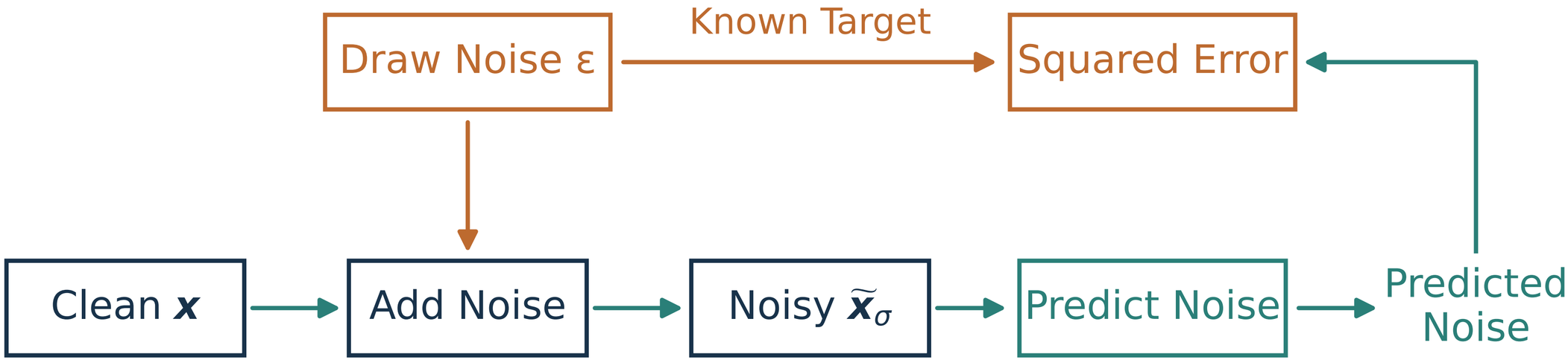
$$\widehat{\boldsymbol{x}} = \widetilde{\boldsymbol{x}}_\sigma - \sigma\epsilon_{\theta,\sigma}(\widetilde{\boldsymbol{x}}_\sigma), \qquad \hat{\boldsymbol{\epsilon}} = \frac{\widetilde{\boldsymbol{x}}_\sigma - D_{\theta,\sigma}(\widetilde{\boldsymbol{x}}_\sigma)}{\sigma}.$$

The squared errors differ by a constant:

$$\|\boldsymbol{x} - \widehat{\boldsymbol{x}}\|^2 = \sigma^2 \|\boldsymbol{\epsilon} - \hat{\boldsymbol{\epsilon}}\|^2.$$

Same prediction problem; a different output parameterization.

The noise we draw becomes a supervised target



$$J(\theta) = \mathbb{E}_{\mathbf{x},\boldsymbol{\epsilon}} \left[\|\boldsymbol{\epsilon} - \epsilon_{\theta,\sigma}(\boldsymbol{x} + \sigma\boldsymbol{\epsilon})\|^2 \right].$$

No noise labels need to be collected: **we constructed the corruption**.

One minibatch update learns noise prediction at one level

Fix $\sigma > 0$. For each batch of B clean images:

Step	Computation, $i = 1, \dots, B$
Draw noise	$\boldsymbol{\epsilon}_i \sim \mathcal{N}(\mathbf{0}, I)$ independently
Corrupt	$\widetilde{\boldsymbol{x}}_{\sigma,i} = \boldsymbol{x}_i + \sigma\boldsymbol{\epsilon}_i$
Predict	$\hat{\boldsymbol{\epsilon}}_i = \epsilon_{\theta,\sigma}(\widetilde{\boldsymbol{x}}_{\sigma,i})$
Average loss	$\widehat{J} = \frac{1}{B}\sum_i \lVert\boldsymbol{\epsilon}_i - \hat{\boldsymbol{\epsilon}}_i\rVert^2$
Update	$\theta \leftarrow \theta - \eta\nabla_\theta \widehat{J}$

Can this one denoiser also turn random noise into a clean image?

From denoising to noisy scores

What the predictor learns—and why one noise level is insufficient

Denoising learns the score of the noisy distribution

$$\widetilde{\boldsymbol{x}}_\sigma = \boldsymbol{x} + \sigma\boldsymbol{\epsilon}, \qquad p_\sigma = p_{\text{data}} * \mathcal{N}(\mathbf{0}, \sigma^2 I).$$

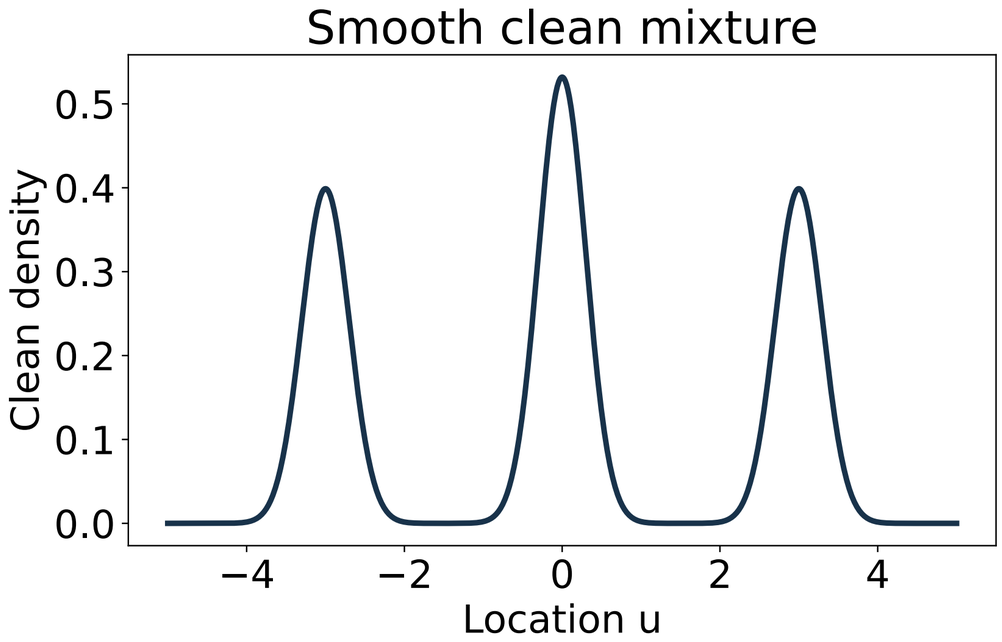
$*$ denotes **convolution**: adding independent Gaussian noise smooths the clean-data distribution.

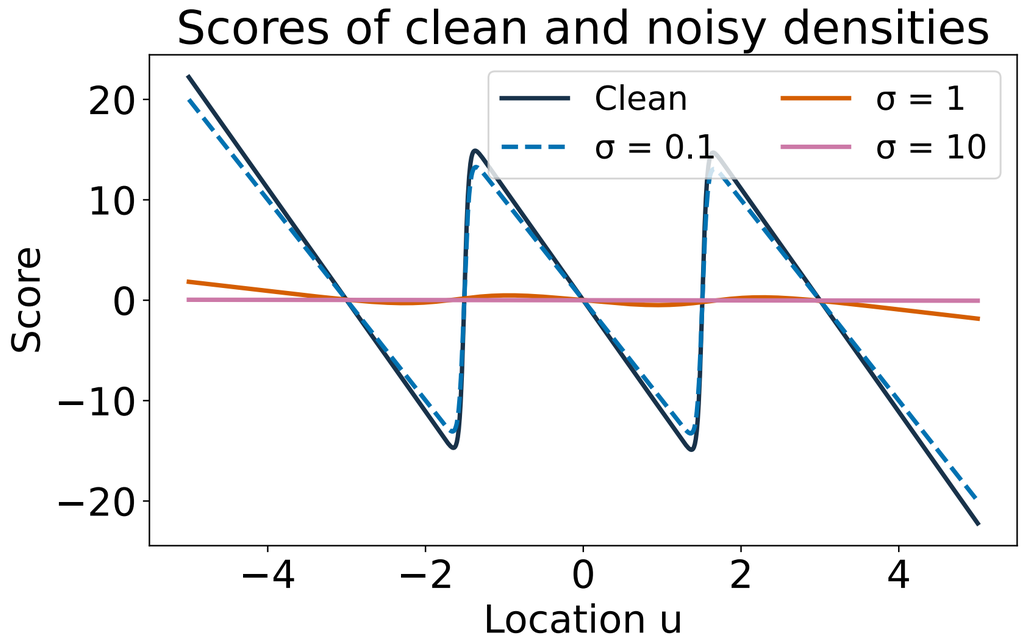
At the noisy input $\widetilde{\boldsymbol{x}}_\sigma$, the score points toward increasing **noisy-data log density**:

$$s_\sigma(\widetilde{\boldsymbol{x}}_\sigma) = \nabla_{\widetilde{\boldsymbol{x}}_\sigma} \log p_\sigma(\widetilde{\boldsymbol{x}}_\sigma) = -\frac{\epsilon^*_\sigma(\widetilde{\boldsymbol{x}}_\sigma)}{\sigma} \quad \approx \quad -\frac{\epsilon_{\theta,\sigma}(\widetilde{\boldsymbol{x}}_\sigma)}{\sigma}.$$

Exact for the ideal squared-error predictor; approximate for the trained network.

Small noise approaches the clean score where that score exists

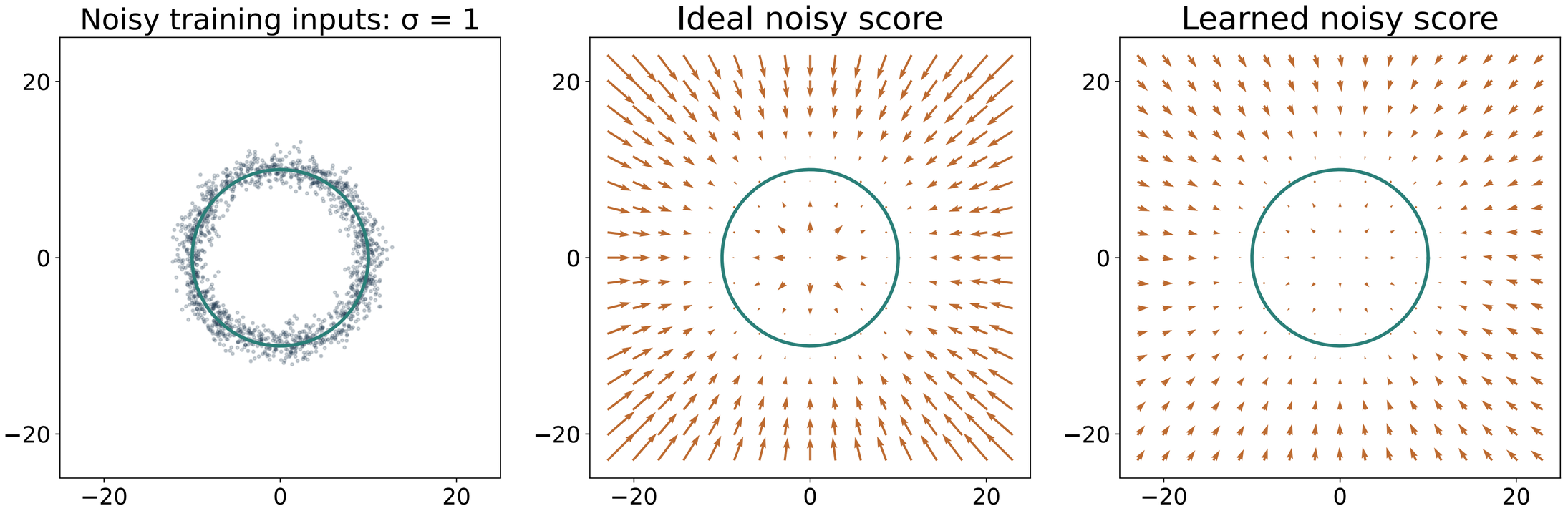




For a smooth positive clean density, under suitable regularity: $\nabla \log p_\sigma(\widetilde{\boldsymbol{x}}_\sigma) \to \nabla \log p_{\text{data}}(\widetilde{\boldsymbol{x}}_\sigma)$ as $\sigma \to 0$.

Small-noise training leaves poorly constrained scores away from data

Clean data: uniform on a radius-10 circle. Train a small MLP at $\sigma = 1$.



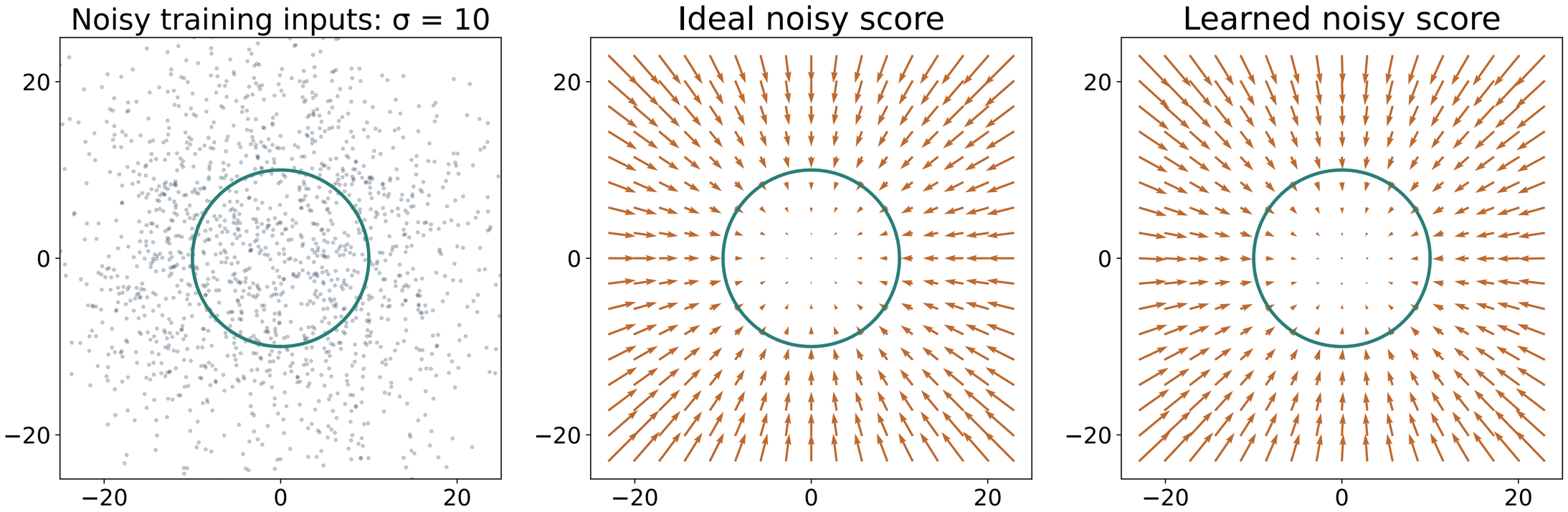
Relative score error: ring band 6% • interior 60% • exterior 53%

Arrow length = score magnitude × 0.2 (both panels) • Teal circle: clean data • Seed 5720

Measured failure here: wrong magnitudes off the ring; directions mostly remain radial.

Large noise improves coverage but changes the score being learned

Same architecture and training budget, now $\sigma = 10$.



Relative score error: ring band 12% • interior 29% • exterior 4%

Arrow length = score magnitude × 20 (both panels) • Teal circle: clean data • Seed 5720

A good estimate of $\nabla \log p_{10}$ is still not the clean-data score.

What would perfect single-level sampling produce?

Suppose the $\sigma = 10$ score is exact and the sampler correctly targets its density.

What would more accurate sampling recover?

A. The clean radius-10 circle

B. The broad noisy distribution p_{10}

C. One conditional-mean point

B. Better approximation cannot change the distribution the score describes.

Learning across noise levels

Broad coverage first; clean-data detail at lower noise

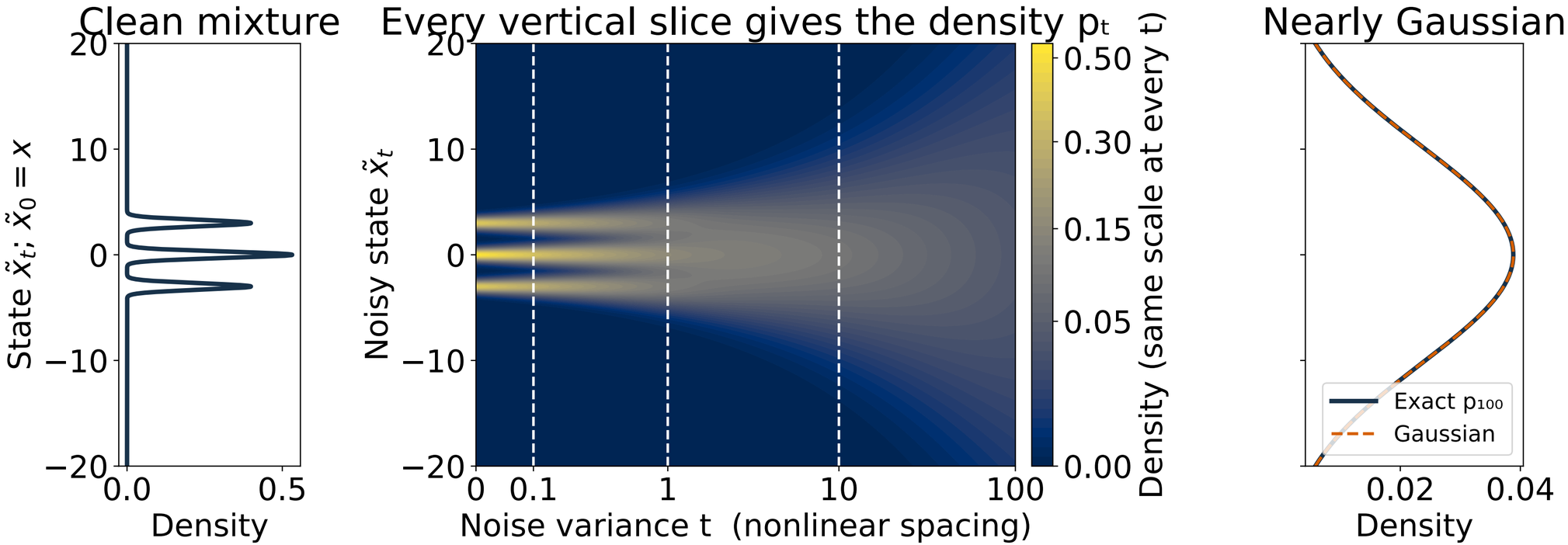
Multiple noise levels connect broad coverage to fine detail

Noisy scores → discrete noise levels → continuous SDE/ODE sampling

Now t denotes noise variance: $\sigma_t = \sqrt{t}, \widetilde{\boldsymbol{x}}_t = \boldsymbol{x} + \sqrt{t}\,\boldsymbol{\epsilon}$, and $\widetilde{\boldsymbol{x}}_0 = \boldsymbol{x}$.

1. Large noise: learn directions across a broad region.
2. Intermediate noise: change the target toward cleaner structure.
3. Small noise: resolve detail near the clean data.

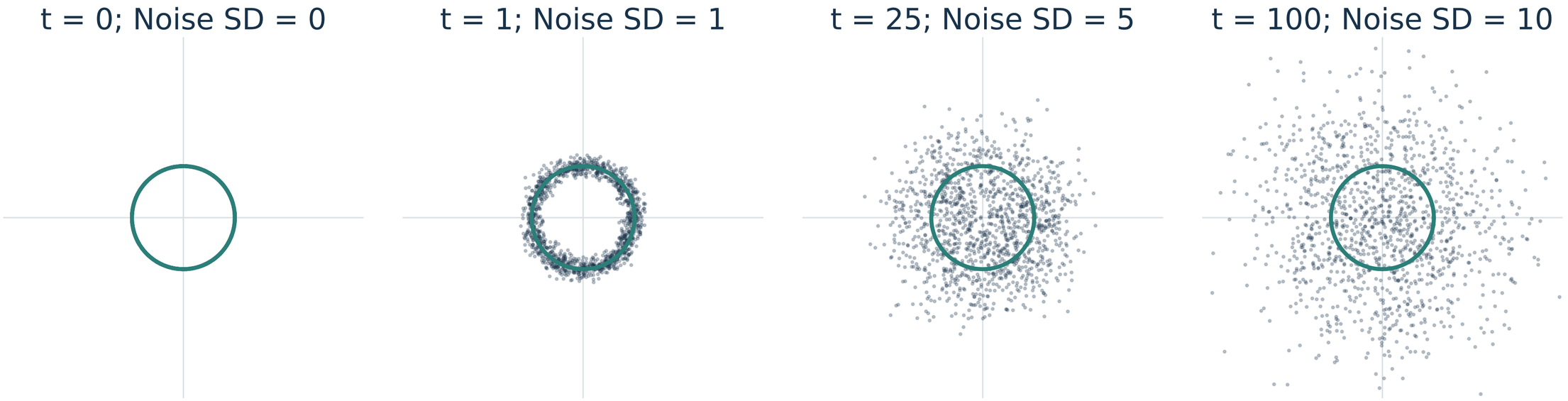
Every time slice is a distribution over possible noisy samples



$$p_t(\tilde{x}_t) = \sum_{j=1}^{3} w_j \, \mathcal{N}(\tilde{x}_t; \mu_j, \, 0.09 + t), \qquad (\mu_1, \mu_2, \mu_3) = (-3, 0, 3).$$

Independent standard-normal steps make noise variance add

$$\widetilde{\boldsymbol{x}}_t = \widetilde{\boldsymbol{x}}_{t-1} + \boldsymbol{\eta}_t, \qquad \boldsymbol{\eta}_t \overset{\text{iid}}{\sim} \mathcal{N}(\mathbf{0}, I).$$



Same Axis Scale in Every Panel; Each Step Adds Independent N(0, I) Noise

After t steps: **noise variance t; noise standard deviation $\sqrt{t}$.**

Any noisy training input needs only one Gaussian draw

$$p_t(\widetilde{\mathbf{x}}_t \mid \boldsymbol{x}) = \mathcal{N}(\widetilde{\mathbf{x}}_t; \boldsymbol{x}, tI), \qquad \widetilde{\boldsymbol{x}}_t = \boldsymbol{x} + \sqrt{t}\,\boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, I).$$

- Independent Gaussian variances add: $I + \cdots + I = tI$.
- Select t and construct its noisy input directly.
- For large T, the noise dominates: start generation approximately from $\mathcal{N}(\mathbf{0}, TI)$.

One network learns noise prediction at every level

Add the noise level to the input: $\hat{\boldsymbol{\epsilon}}_i = \epsilon_\theta(\widetilde{\boldsymbol{x}}_{t_i}, t_i)$.

Step	Computation for each image in a batch
Draw level and noise	$t_i \sim \mathrm{Uniform}\{1, \dots, T\}, \boldsymbol{\epsilon}_i \sim \mathcal{N}(\mathbf{0}, I)$
Construct input	$\widetilde{\boldsymbol{x}}_{t_i} = \boldsymbol{x}_i + \sqrt{t_i}\,\boldsymbol{\epsilon}_i$
Predict and score	$\widehat{J} = \frac{1}{B}\sum_i \lVert \boldsymbol{\epsilon}_i - \epsilon_\theta(\widetilde{\boldsymbol{x}}_{t_i}, t_i)\rVert^2$
Update shared parameters	$\theta \leftarrow \theta - \eta \nabla_\theta \widehat{J}$

One sampled level per image; the same parameters across levels.

Reverse generation needs small conditional steps, not one clean guess

A clean-image estimate uses the whole predicted noise:

$$\widehat{\boldsymbol{x}} = \widetilde{\boldsymbol{x}}_t - \sqrt{t}\,\epsilon_\theta(\widetilde{\boldsymbol{x}}_t, t).$$

For the additive walk, the **previous-step conditional mean** instead uses

$$\widehat{\boldsymbol{\mu}}_{t-1} = \widetilde{\boldsymbol{x}}_t - \frac{1}{\sqrt{t}}\epsilon_\theta(\widetilde{\boldsymbol{x}}_t, t).$$

Generation follows $T \to T-1 \to \cdots \to 0$ with fixed learned parameters.

A mean is not a full distribution: a reverse sampler must also handle uncertainty.

A stochastic reverse step denoises, then adds fresh noise

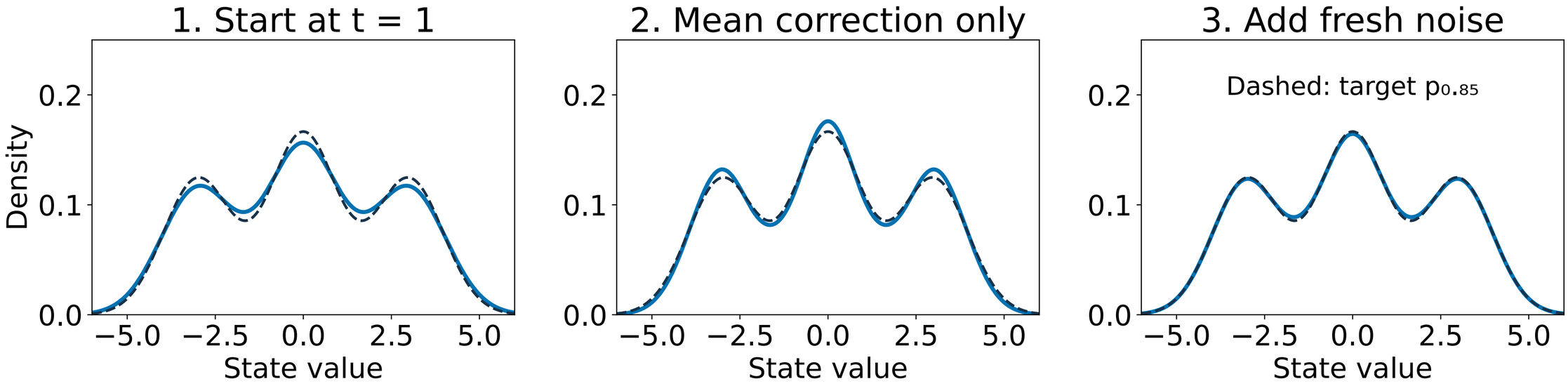
Move from current noise variance t to a lower variance $s \geq 0$. The **step size** is $h = t - s > 0$: how much we reduce the noise variance.

A **DDPM-style step in our additive units** is

$$\tilde{x}_s = \underbrace{\tilde{x}_t - \frac{h}{\sqrt{t}}\epsilon_\theta(\tilde{x}_t, t)}_{m\text{: previous-level mean}} + \underbrace{\sqrt{\frac{sh}{t}}\,\xi}_{\text{fresh noise}}, \qquad \xi \sim \mathcal{N}(0,1).$$

- The correction uses h/t of the predicted clean-data correction.
- For small h relative to t, the added variance is approximately h.
- This Gaussian step **approximates** the reverse conditional distribution.

Mean correction narrows the distribution; noise spreads it again



Local Gaussian intuition: variance $v \to v - 2h \to v - h$ (first order in h)

“Denoise two, add back one” describes local variance change—not a time reversal.

Diffusion as a continuous process

From Gaussian steps to SDEs and continuous-time training

Continuous noising is the small-step limit of the Gaussian walk

Discrete diffusion → continuous SDE → probability-flow ODE → solver choices

$$d\widetilde{\boldsymbol{x}}_t = d\mathbf{w}_t, \qquad \widetilde{\boldsymbol{x}}_0 = \boldsymbol{x}.$$

- An **SDE** adds random increments: $\Delta\mathbf{w} \sim \mathcal{N}(\mathbf{0}, hI)$ over an interval h.
- Our time label is noise variance: $\sigma(t) = \sqrt{t}$.
- Its marginal remains $\widetilde{\boldsymbol{x}}_t = \boldsymbol{x} + \sqrt{t}\,\boldsymbol{\epsilon}$ in distribution.

Continuous-time training still needs one noise draw per example

Draw $t \sim \rho$ on $[t_{\min}, T]$, $t_{\min} > 0$, then form $\widetilde{\boldsymbol{x}}_t = \boldsymbol{x} + \sqrt{t}\,\boldsymbol{\epsilon}$.

$$\min_{\theta} \; \mathbb{E}\left[\lambda(t)\|\boldsymbol{\epsilon} - \epsilon_\theta(\widetilde{\boldsymbol{x}}_t, t)\|^2\right], \qquad s_\theta(\widetilde{\boldsymbol{x}}_t, t) = -\frac{\epsilon_\theta(\widetilde{\boldsymbol{x}}_t, t)}{\sqrt{t}}.$$

No simulated SDE trajectory and no SDE solver are required for this training loss.

The reverse SDE repeatedly combines score drift and random increments

Let generation time $\tau = T - t$ increase while noise variance t decreases:

$$d\widetilde{\boldsymbol{x}}_{T-\tau} = s_\theta(\widetilde{\boldsymbol{x}}_{T-\tau}, T-\tau)\, d\tau + d\overline{\mathbf{w}}_\tau.$$

A small numerical step is

$$\widetilde{\boldsymbol{x}}_{t-h} \approx \widetilde{\boldsymbol{x}}_t + h\, s_\theta(\widetilde{\boldsymbol{x}}_t, t) + \sqrt{h}\, \boldsymbol{\xi}.$$

Feed the new state into the next step; keep the network parameters fixed.

What if we delete the fresh noise from this sampler?

> **Keep the random initialization and the same mean correction.**
>
> **What should we expect if every ξ is set to zero?**
>
> **A. The same distribution, computed faster**
>
> **B. A generally different output distribution**
>
> **C. The same output on every run**

B. A valid deterministic sampler also changes the correction.

One model, different sample paths

ODE and SDE samplers can share the same ideal marginals

The probability-flow ODE transports the same density without fresh noise

Using the **same score predictor** and increasing reverse clock τ:

$$\frac{d\widetilde{\boldsymbol{x}}_{T-\tau}}{d\tau} = \frac{1}{2}\, s_\theta(\widetilde{\boldsymbol{x}}_{T-\tau}, T-\tau).$$

- An **ODE** gives a deterministic trajectory once the initial sample is drawn.
- Half the additive-process drift replaces the SDE's drift-plus-noise balance.
- Exact score + exact initial distribution + exact integration ⇒ **same marginals** p_t.

We can vary stochasticity without retraining the score model

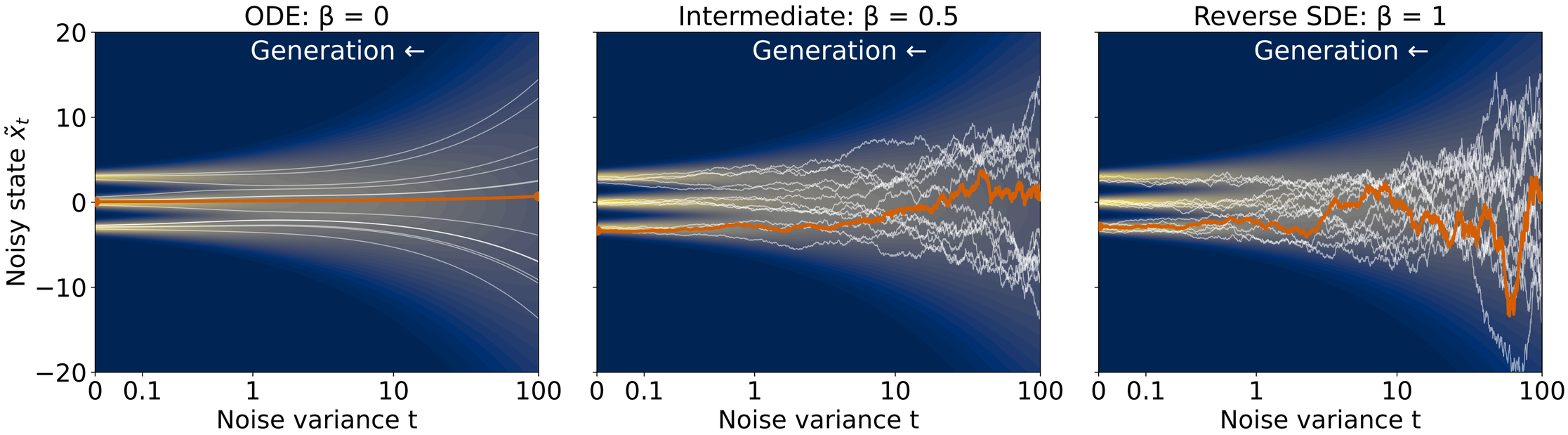
For $0 \leq \beta \leq 1$, one family in our reverse clock is

$$d\widetilde{\boldsymbol{x}}_{T-\tau} = \frac{1+\beta^2}{2}\, s_\theta(\widetilde{\boldsymbol{x}}_{T-\tau}, T-\tau)\, d\tau + \beta\, d\overline{\mathbf{w}}_\tau.$$

$\beta = 0$	$0 < \beta < 1$	$\beta = 1$
Probability-flow ODE	Intermediate stochastic sampler	Reverse diffusion SDE

Change the drift together with the noise amplitude to preserve the ideal marginals.

Different sample paths can pass through the same marginal distributions

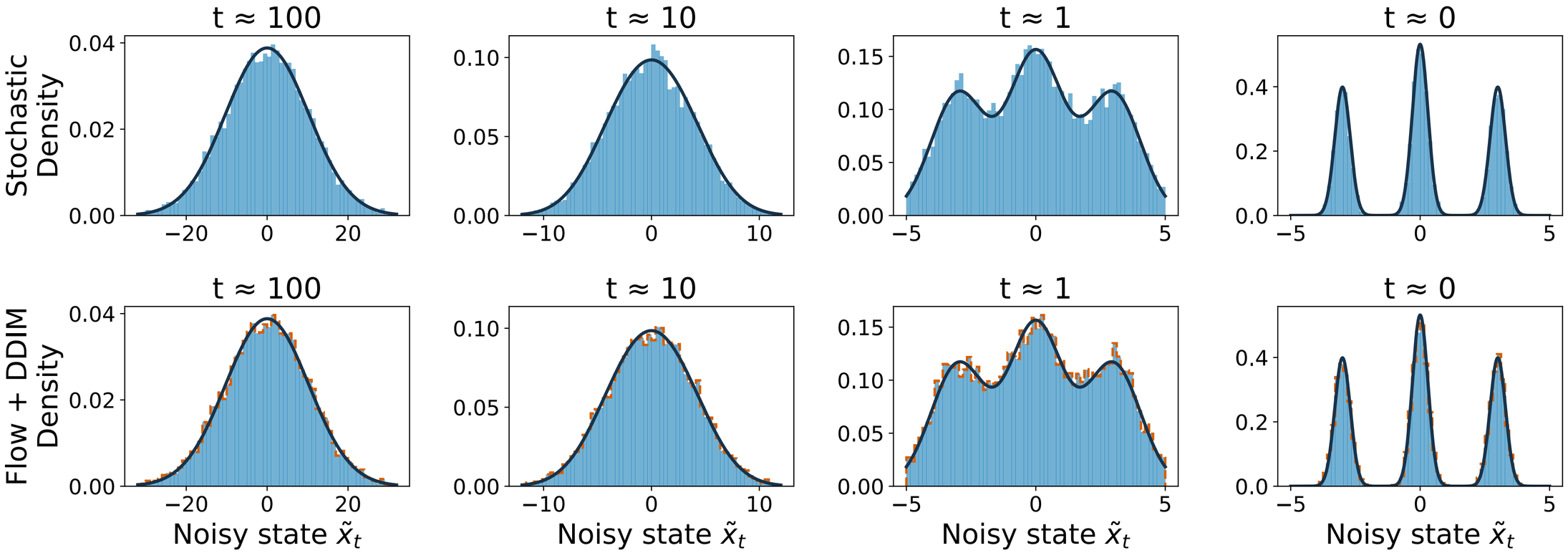


Nonlinear time spacing • Same density scale and matched starting states

A vertical slice describes a population of samples, not one trajectory.

Many sampled paths recover the target density at each time

12,000 samples each • Black: p_t • Blue: sampled states • Dashed orange: DDIM



Compare the distributions at matched times—not matched outputs from one seed.

DDIM reuses the predicted noise direction without drawing fresh noise

First predict clean data, then retain the noise appropriate for $s < t$:

$$\hat{x} = \tilde{x}_t - \sqrt{t}\,\epsilon_\theta(\tilde{x}_t, t), \qquad \tilde{x}_s = \hat{x} + \sqrt{s}\,\epsilon_\theta(\tilde{x}_t, t).$$

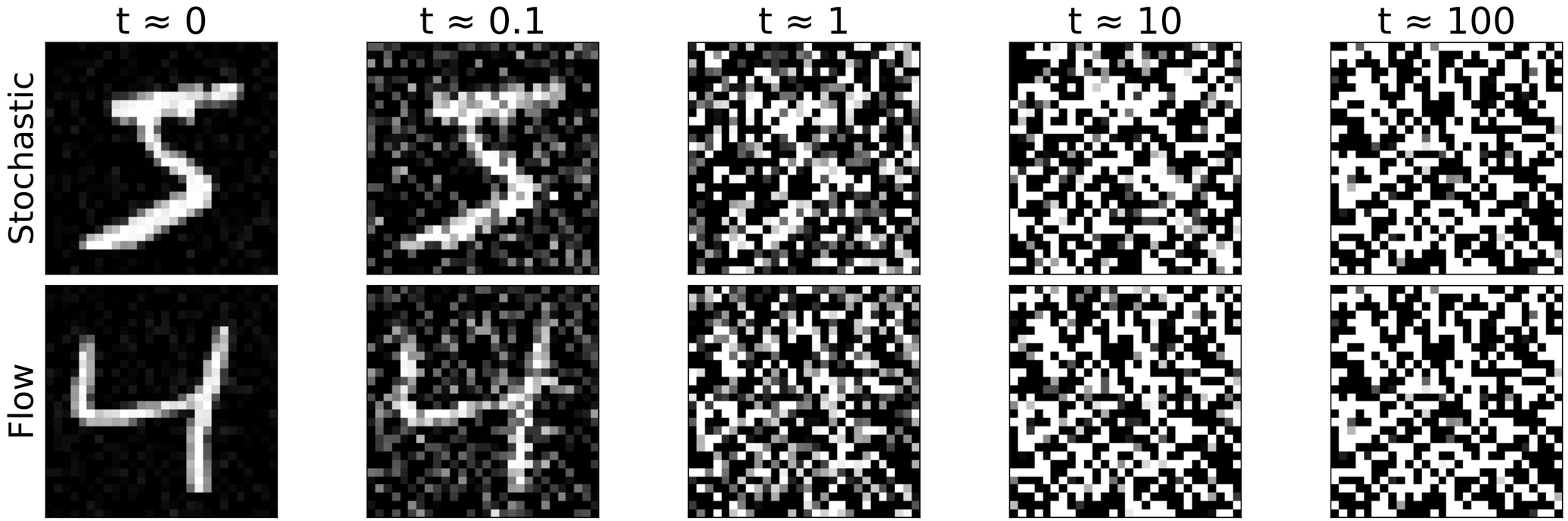
Therefore

$$\tilde{x}_s = \tilde{x}_t + (\sqrt{s} - \sqrt{t})\epsilon_\theta(\tilde{x}_t, t) \;\approx\; \tilde{x}_t - \frac{h}{2\sqrt{t}}\epsilon_\theta(\tilde{x}_t, t).$$

Deterministic DDIM approaches probability flow as steps become small.

An image follows one path through a distribution over whole images

Generation ← • Same initial noisy image, same fixed analytic image predictor



Toy distribution around 3 MNIST images; not a trained image model. Display clips pixel values to [0, 1].

The 1D value $\tilde{x}_t$ becomes an image array $\widetilde{\boldsymbol{x}}_t$; p_t **is not a pixel histogram**.

Choosing a numerical sampler

Solver accuracy, computation and downstream quality

Numerical integration turns the learned field into a practical sampler

Write the reverse ODE drift as $b(\widetilde{\boldsymbol{x}}_t, t) = \frac{1}{2} s_\theta(\widetilde{\boldsymbol{x}}_t, t)$.

Euler proposal:

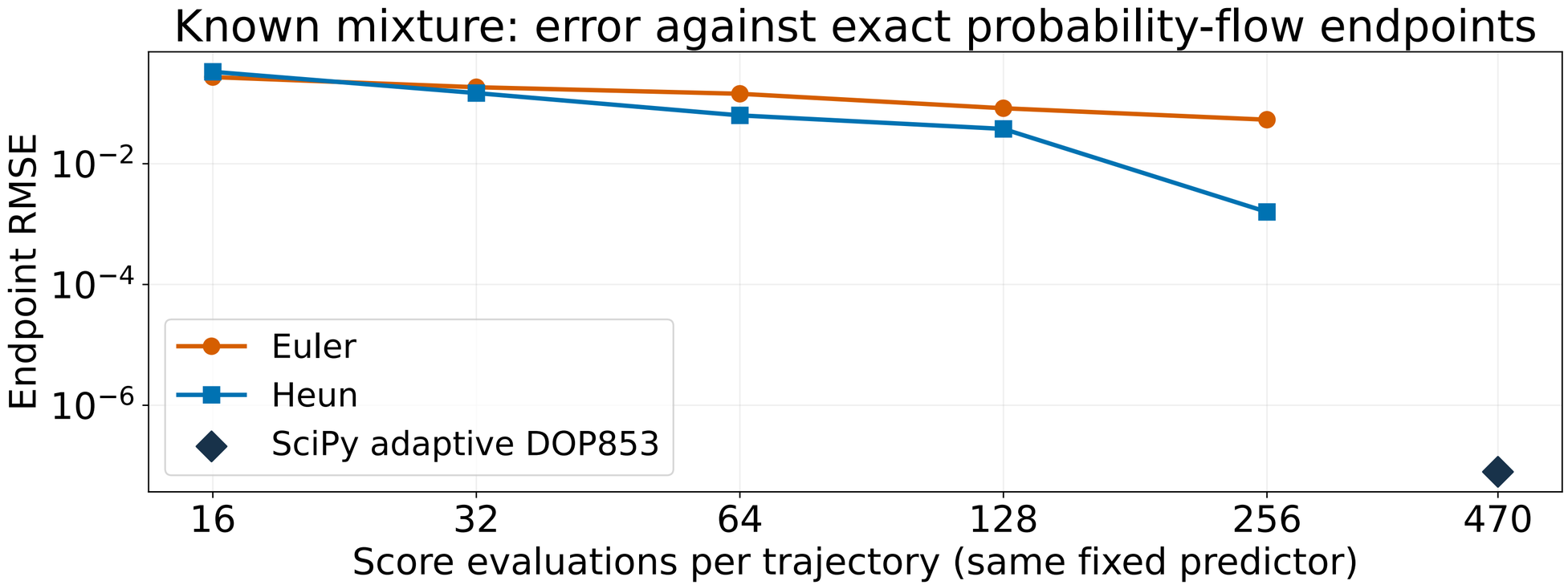
$$\widetilde{\boldsymbol{x}}^{\mathrm{E}}_{t-h} = \widetilde{\boldsymbol{x}}_t + h\, b(\widetilde{\boldsymbol{x}}_t, t).$$

Heun correction:

$$\widetilde{\boldsymbol{x}}_{t-h} = \widetilde{\boldsymbol{x}}_t + \frac{h}{2}\left[b(\widetilde{\boldsymbol{x}}_t, t) + b(\widetilde{\boldsymbol{x}}^{\mathrm{E}}_{t-h}, t-h)\right].$$

Evaluate two slopes instead of one; compare cost in network evaluations.

A better solver can improve sampling with exactly the same predictor



This isolates solver error. Image quality must also be measured downstream.

Training and sampling are separate design choices with shared constraints

Learn the noisy score	Turn that score into samples
Data, architecture and conditioning	ODE, SDE or intermediate stochasticity
Noise-level distribution and loss weighting	Noise grid and integration variable
Prediction target and input/output scaling	Solver order, evaluation budget and endpoint

Hold training fixed to compare samplers; evaluate final quality, diversity and cost.

NVIDIA: Elucidating the Design Space of Diffusion-Based Generative Models — Karras et al., NeurIPS 2022

Conditioning makes denoising depend on the requested output

For an image paired with a condition c:

$$\widetilde{\boldsymbol{x}}_t = \boldsymbol{x} + \sqrt{t}\,\boldsymbol{\epsilon}, \qquad \hat{\boldsymbol{\epsilon}} = \epsilon_\theta(\widetilde{\boldsymbol{x}}_t, t, c).$$

- **Train:** retain c with each image while learning noise prediction.
- **Generate:** supply the same request at every reverse step.
- A class label, text embedding, or layout can provide c; **request adherence needs testing**.

Latent diffusion performs the repeated steps in a compressed image space

Training: Construct Noisy Codes



Generation: Denoise Codes, Then Decode



Encode for training; repeatedly denoise codes; decode once after generation.

The codec and denoiser create different sources of error

Check	What it isolates
Real image → encode → decode	Detail lost by the codec
Noise → latent denoising → decode	End-to-end generation quality and diversity
Repeat requests across seeds	Consistency and condition adherence

Smaller latent arrays can reduce repeated computation, **with a compression tradeoff**.

Generative families differ in how a sample is produced

Model	Generation procedure	Main computation to budget
VAE	Sample a latent code, then decode	Decoder evaluation
Diffusion	Start with noise, then follow reverse steps	Repeated denoiser evaluations
Autoregressive	Sample the next symbol conditioned on the prefix	Sequential symbol-generation steps

A vanilla AE or fixed-level DAE **does not specify a generator by itself**.

A plausible image still needs evidence for its intended use

Suppose the generator illustrates specified objects in a teaching worksheet.

Intended value	Evidence to collect
Correct requested objects and counts	Independent request set; structured human checks
Useful variety without broken examples	Multiple seeds per request; inspect failures by slice
Practical classroom use	Readability at final size, generation time, and review effort

Next: turn these checks into an evaluation protocol and compare models fairly.