上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Lecture 26: Diffusion Models

Tao Huang

John Hopcroft Center, School of Computer Science, Shanghai Jiao Tong University

https://taohuang.info/cs3317

https://oc.sjtu.edu.cn/courses/89538

AI tools assisted in generating some figures in these slides. All such content has been reviewed, and the instructor is responsible for its accuracy.

Where We Left Off: The Generative Trilemma

L23 Autoregressive

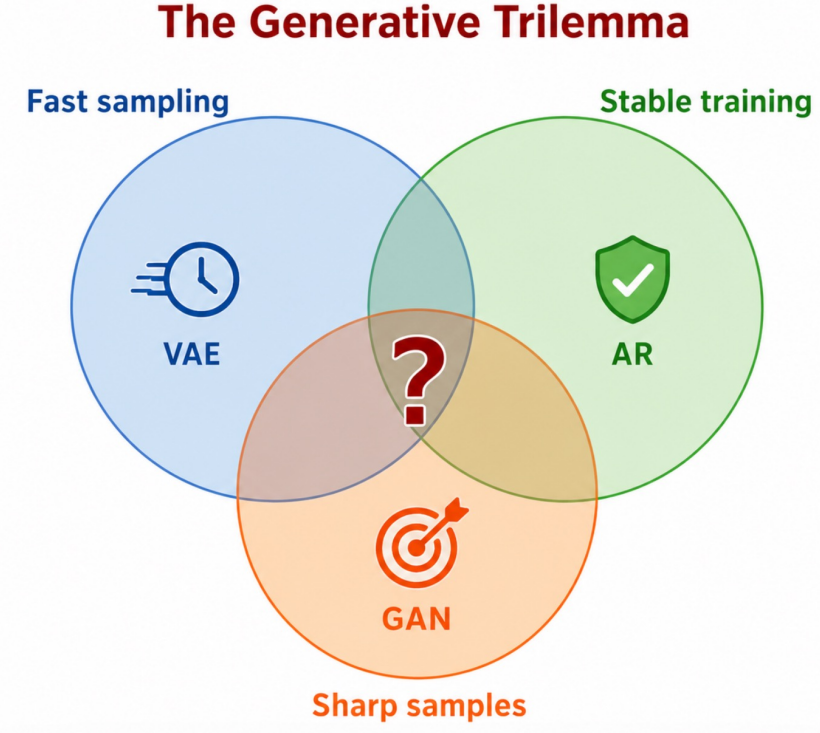
✓ exact likelihood ✓ stable training
✗ slow sampling (N forward passes)

L24 VAE

✓ one-shot sampling ✓ likelihood (lower bound)
✗ blurry samples

L25 GAN

✓ one-shot sampling ✓ sharp samples
✗ unstable ✗ no likelihood ✗ mode collapse



Objectives

By the end of this lecture, you should be able to:

- Derive the forward noising process — and explain why it is a fixed (non-learned) Markov chain.
- Analyze the reverse process as a denoiser, and connect it to the score function $\nabla_x \log p(x)$.
- Connect the diffusion training objective to the hierarchical VAE ELBO (Ho et al. 2020).
- Evaluate the trade-offs vs. autoregressive / VAE / GAN — and explain why diffusion broke the trilemma.
- Recognize flow matching as the modern unification used in SD3, FLUX, Movie Gen.

Outline

Part I.

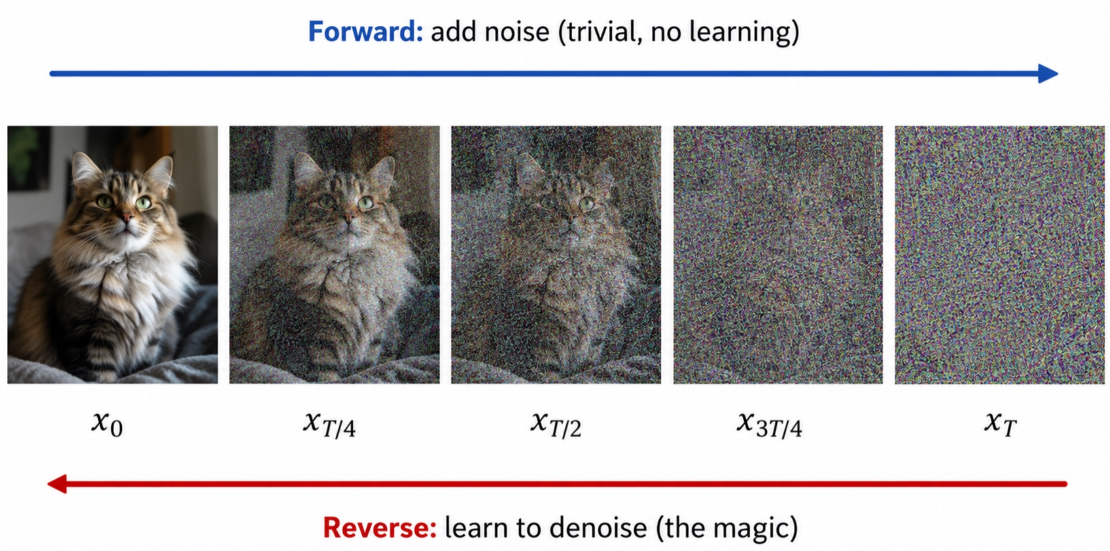
Forward — destroy structure with noise (the easy direction).

Part II.

Reverse — learn to undo it (the hard direction, where the magic is).

Part III.

Why it works — score matching, ELBO connection, sampling, scaling.



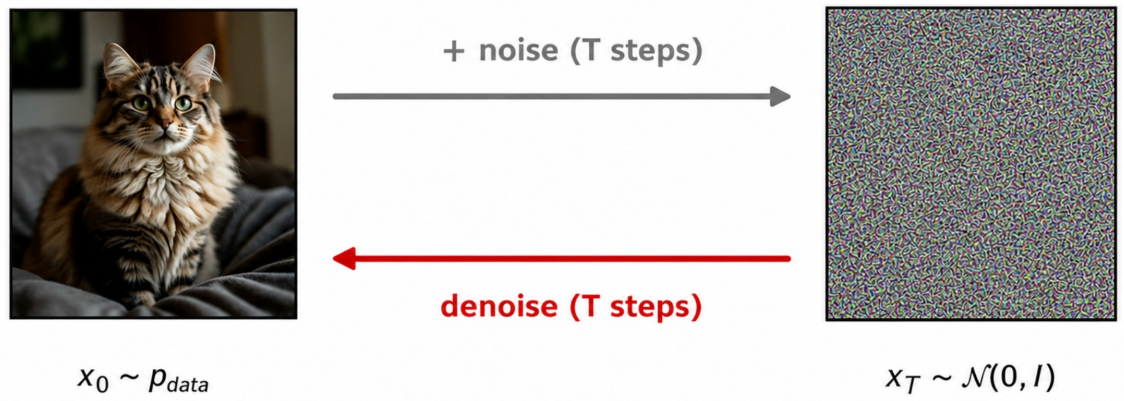
1. Forward — Destroying Structure on Purpose

A Strange Idea: Generation Backwards

Every prior model tried to learn the data distribution directly.

Diffusion's bet: don't try. Instead —

- destroy it slowly with noise,
- then learn to reverse the destruction.



If we can undo each tiny destruction step, we can walk Gaussian noise back to a sample from p(x).

Why would adding noise help?

The Forward Markov Chain

Define a chain $x_0 \to x_1 \to \cdots \to x_T$ where $x_0 \sim p_{data}$, each step adds Gaussian noise:

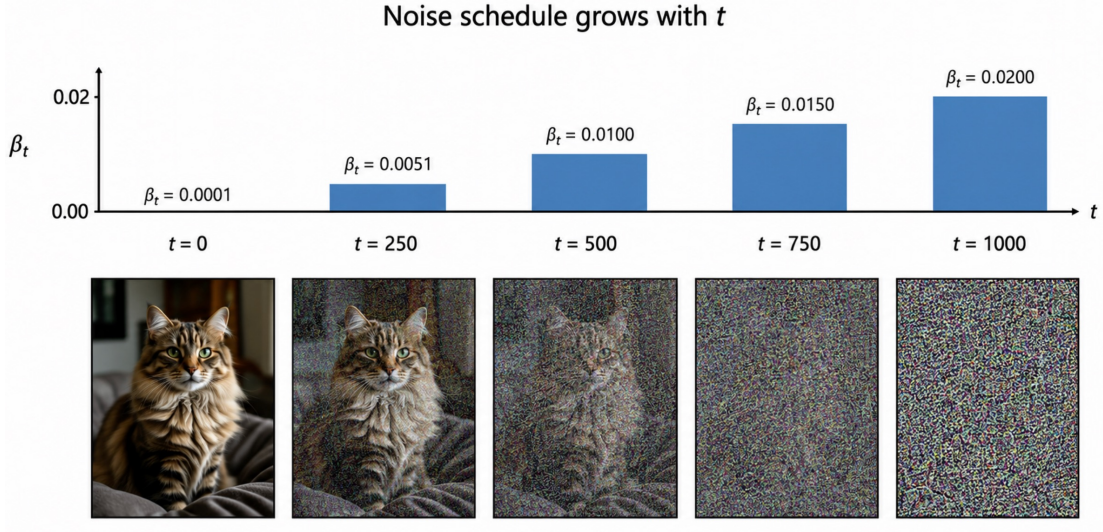
$$q(x_t \mid x_{t-1}) = \mathcal{N}\left(x_t;\ \sqrt{1-\beta_t}\, x_{t-1},\ \beta_t I\right)$$

- $\beta_t \in (0,1)$ is the noise schedule (small, increasing).

- **Two crucial properties:**

  (i) fixed — no parameters to learn

  (ii) Gaussian — closed-form everywhere



The “Nice Property”: Skip Ahead in Closed Form

Let $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$. Then:

$$q(x_t \mid x_0) = \mathcal{N}\left(x_t;\ \sqrt{\bar{\alpha}_t}\, x_0,\ (1 - \bar{\alpha}_t) I\right)$$

Equivalently (the reparameterization trick — same form as L24):

$$x_t = \sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1 - \bar{\alpha}_t}\ \varepsilon, \quad \varepsilon \sim \mathcal{N}(0, I)$$

No need to simulate t steps — sample x_t directly in O(1).

This is what makes training feasible (we will see why on slide 14).

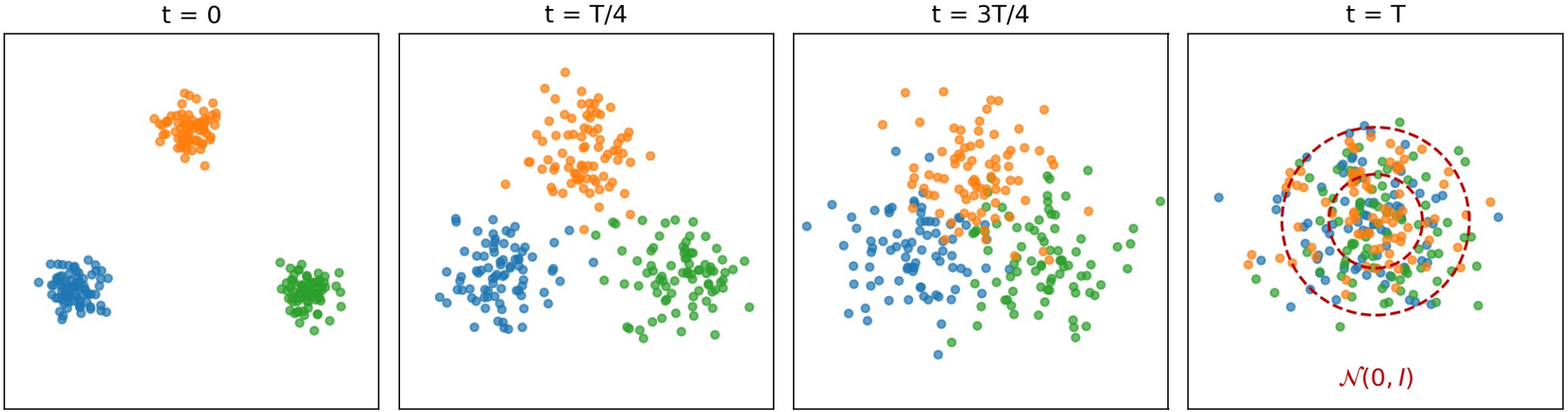
Where Do We End Up?

Take T large enough and $\bar{\alpha}_T \to 0$, so $q(x_T) \to \mathcal{N}(0, I)$.

The forward process always ends at standard Gaussian, regardless of x_0.

This is the prior we will sample from at generation time.

The forward process erases everything you knew about x_0.



🧠 Think: Why Add Noise Slowly?

We could just set $q(x_t \mid x_0) = \mathcal{N}(0, I)$ directly. Why bother with T = 1000 small steps?

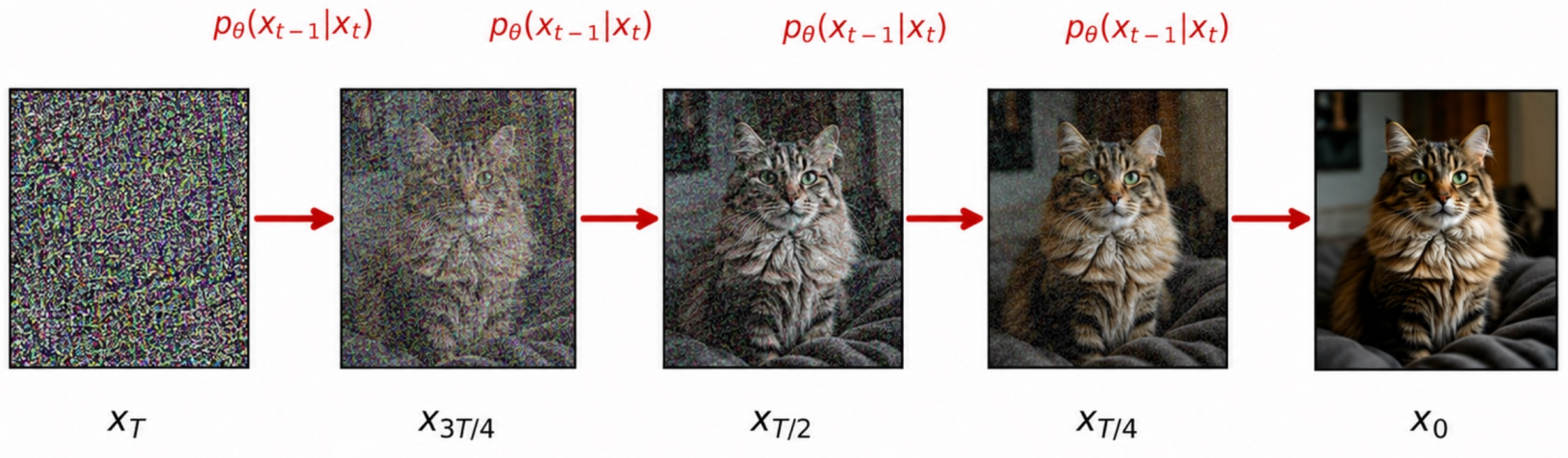
1. What property of the reverse process would we lose if T = 1?

2. What is the relationship between step size β_t and how easy $q(x_{t-1} \mid x_t)$ is to model?

2. Reverse — Learning to Undo Destruction

What We Want at Sampling Time

- Start with $x_T \sim \mathcal{N}(0, I)$.
- Repeatedly draw $x_{t-1} \sim p(x_{t-1} \mid x_t)$ for $t = T, T-1, \ldots, 1$.
- Output x_0.
- *The challenge: $p(x_{t-1}|x_t)$ is not something we know — it depends on the data distribution.*



We don't know it. So we'll learn it.

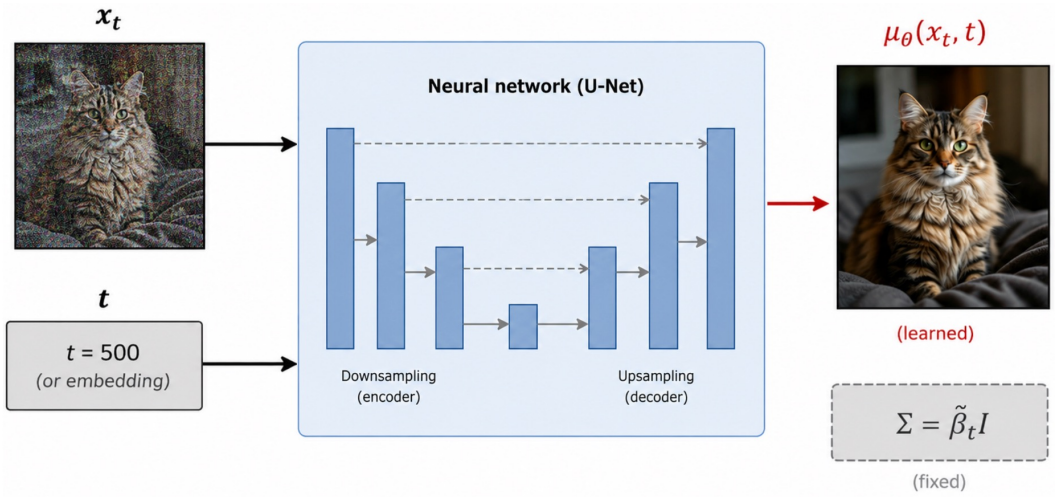
Modeling $p_\theta(x_{t-1} \mid x_t)$

Sohl-Dickstein (2015): when β_t is small, the true reverse $q(x_{t-1} \mid x_t)$ is approximately Gaussian.

So we parameterize:

$$p_\theta(x_{t-1} \mid x_t) = \mathcal{N}(x_{t-1};\ \mu_\theta(x_t, t),\ \Sigma_\theta(x_t, t))$$

- *In DDPM, only the mean μ_θ is learned — Σ is fixed to a function of β_t .*
- A neural network's job is to predict the mean of the Gaussian that takes us one step back.



What Should We Train On?

- Want $p_\theta(x_{t-1} \mid x_t) \approx q(x_{t-1} \mid x_t, x_0)$.
- The true posterior $q(x_{t-1} \mid x_t, x_0)$ is tractably Gaussian (because we conditioned on x_0) :

$$q(x_{t-1} \mid x_t, x_0) = \mathcal{N}(x_{t-1};\ \tilde{\mu}_t(x_t, x_0),\ \tilde{\beta}_t I)$$

where $\tilde{\mu}_t$ has a closed form in terms of x_t, x_0.

- So: train μ_θ to match $\tilde{\mu}_t$ via MSE on means. Sample t uniformly, sample (x_0, ε), compute x_t, apply network, compare.
- *This works. But there is a much cleaner reparameterization coming.*

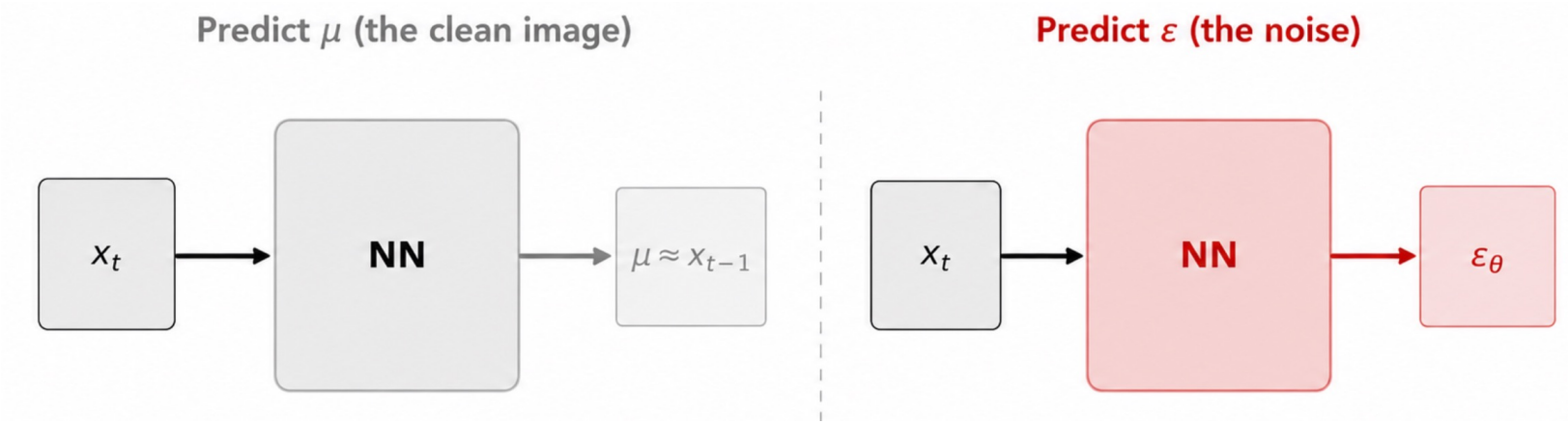
ε-Prediction: Predict the Noise, Not the Mean

Recall: $x_t = \sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1-\bar{\alpha}_t}\, \varepsilon$. Substitute into $\tilde{\mu}_t$ and rearrange (Ho et al. 2020):

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\,\varepsilon_\theta(x_t, t)\right)$$

The network's actual job: predict the noise ε that was added to x_0.

Same model, same loss family — but predicting ε is dramatically easier in practice.



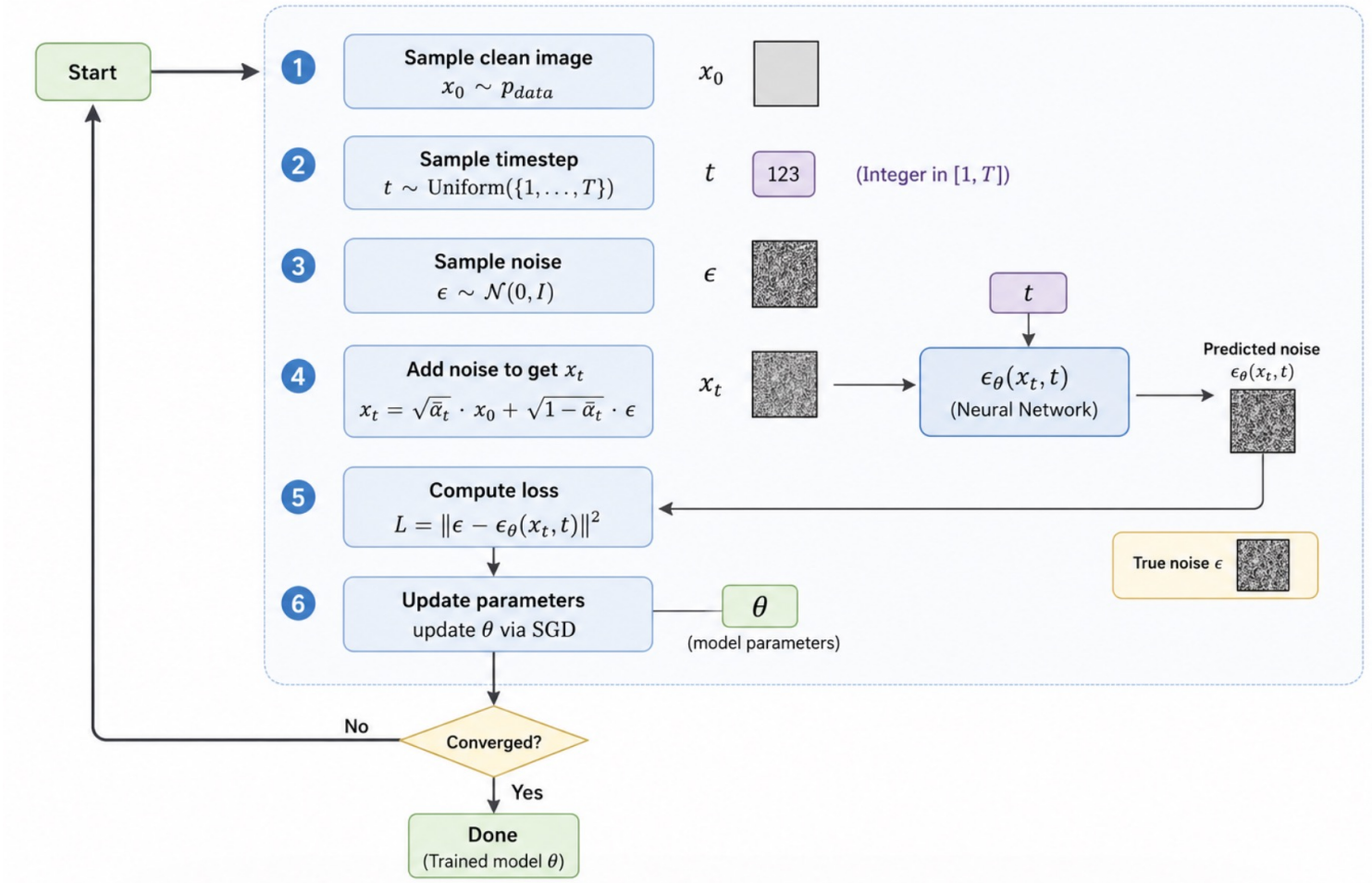
DDPM Training

$$\mathcal{L}_{\text{simple}} = \mathbb{E}_{t,\, x_0,\, \varepsilon}\left[\, \|\varepsilon - \varepsilon_\theta(x_t,\, t)\|^2 \,\right]$$

$$\text{where } x_t = \sqrt{\bar{\alpha}_t} \cdot x_0 + \sqrt{(1 - \bar{\alpha}_t)} \cdot \varepsilon, \quad t \sim Uniform\{1, \ldots, T\}, \quad \varepsilon \sim \mathcal{N}(0, I).$$

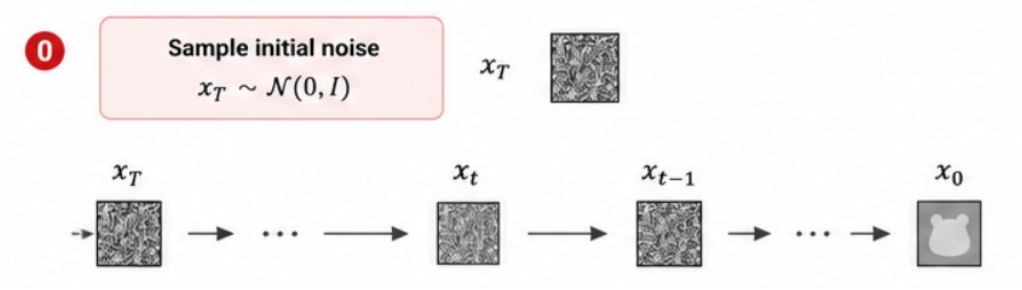
- It is literally MSE between true noise and predicted noise.
- One unified loss handles all timesteps t — no per-t network.
- Time t is fed to the network as a sinusoidal embedding (like positional encoding from L22).

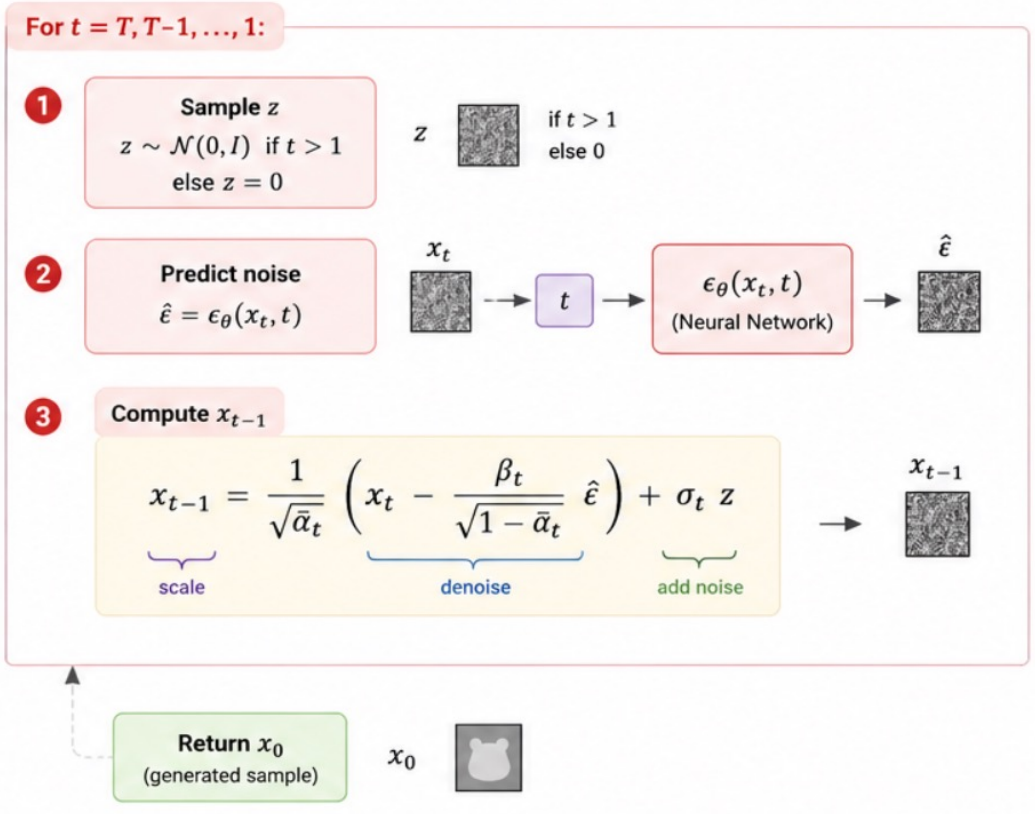
DDPM Training Loop — The Whole Algorithm



No adversarial game. No posterior network. No autoregressive unrolling.
Just denoising.

DDPM Sampling — Walk the Chain Backwards





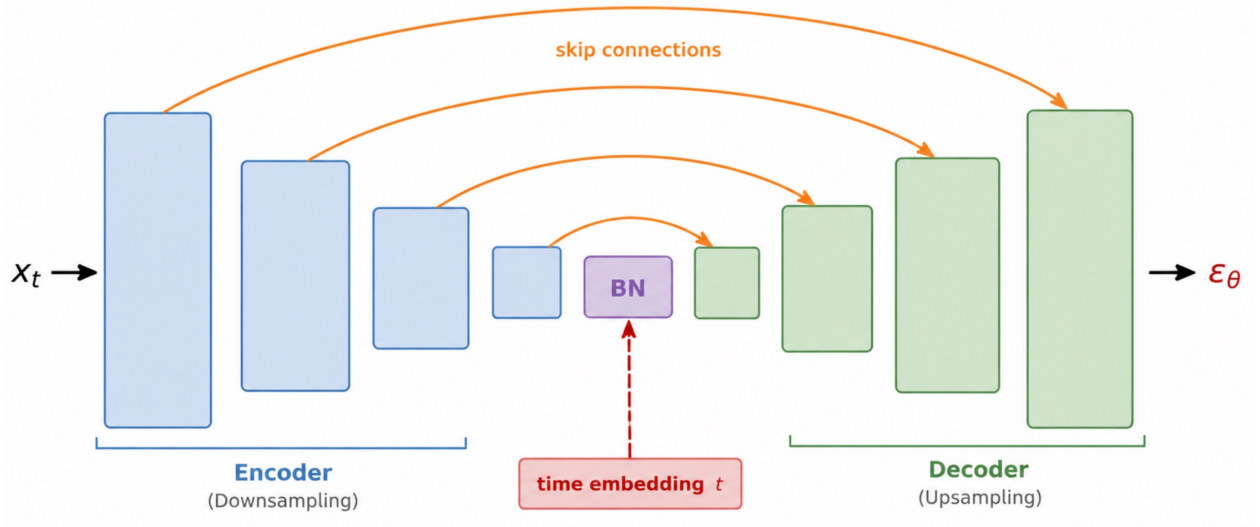
Notation

$\beta_t \in (0,1)$ (noise schedule) | $\alpha_t = 1 - \beta_t$ | $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$ | $\sigma_t = \begin{cases} \sqrt{\beta_t}, & t > 1 \\ 0, & t = 1 \end{cases}$

T = 1000 means 1000 forward passes per sample. GAN does it in 1.

We've gained quality and stability — at the cost of sampling speed. (We'll fix this on slide 27.)

Architecture: The U-Net



- Diffusion is a dense prediction task: input is a noisy image, output is a same-shape noise image.
- U-Net (Ronneberger 2015): encoder–decoder with skip connections between matching resolutions.
- Skip connections are critical — they let high-frequency detail bypass the bottleneck.
- Time t is injected via sinusoidal embedding + MLP, added at each block.

🧠 Think: Why Predict Noise Instead of Image?

Both are mathematically equivalent (slide 15). Why is ε-prediction dramatically better in practice?

1. At t = T, what does x_0-prediction have to output? What about ε-prediction?

2. Think about loss scale. As t varies, does $\|x_0\|$ stay constant? Does $\|\varepsilon\|$?

3. Why It Works — Score Matching, ELBO, Sampling

Diffusion as a Hierarchical VAE

- Recall L24: VAE maximizes the ELBO on log p(x).
- Diffusion is a VAE with T latent variables (x_1, …, x_T), and a fixed encoder q (no learned parameters!).

The full ELBO:

$$\log p_\theta(x_0) \geq \mathbb{E}_q[\log p_\theta(x_0 \mid x_1) - \sum_{t=2}^{T} D_{\mathrm{KL}}(q(x_{t-1} \mid x_t, x_0) \parallel p_\theta(x_{t-1} \mid x_t)) - D_{\mathrm{KL}}(q(x_T \mid x_0) \parallel p(x_T))]$$

Each KL term is between two Gaussians — closed form. The training trick is structural.

From ELBO to L_{simple}

- Each KL term reduces to (Ho et al. 2020):

$$D_{\mathrm{KL}}(q(x_{t-1}|x_t, x_0) \,\|\, p_\theta(x_{t-1}|x_t)) = \mathbb{E}\left[w_t \,\|\varepsilon - \varepsilon_\theta(x_t, t)\|^2 \right]$$

- The proper VLB uses the weights w_t. Ho et al. (2020) drop them — set $w_t = 1$. That gives L_simple.
- Empirically: dropping the weights improves sample quality.
- Interpretation: $w_t \propto 1 / \mathrm{SNR}(t)$ down-weights small-noise steps; setting $w_t = 1$ reweights training toward harder, noisier steps.

Variational lower bound is the theoretical objective. L_simple is the empirical objective. They differ — and the simpler one wins.

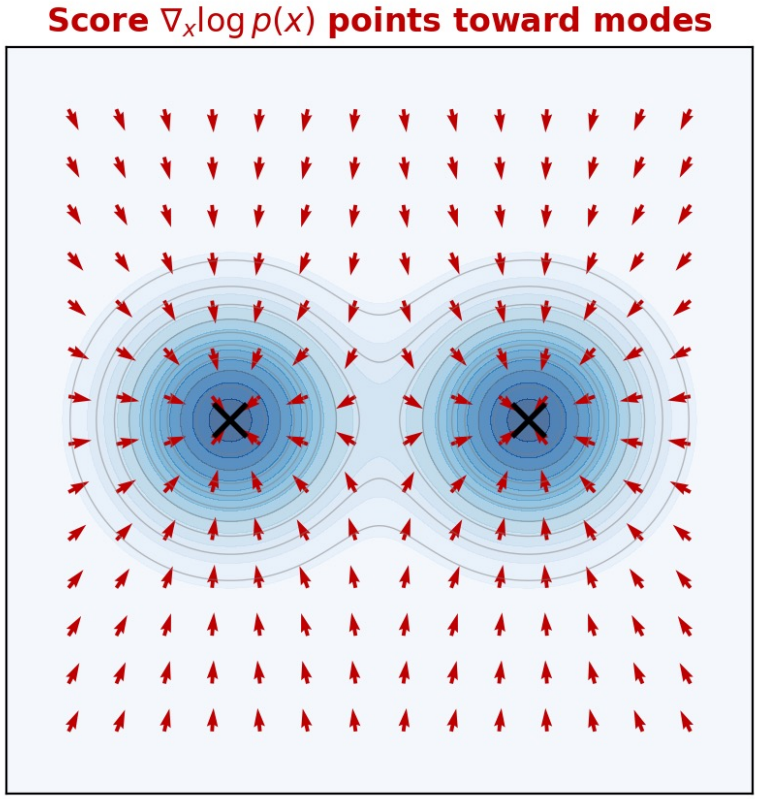
ε-Prediction Is Score Estimation

- Score function: $s(x) = \nabla_x \log p(x)$ — the direction of fastest density increase.
- For $q(x_t \mid x_0) = \mathcal{N}\left(\sqrt{\bar{\alpha}_t}\, x_0, (1-\bar{\alpha}_t)I\right)$:

$$\nabla_{x_t} \log q(x_t \mid x_0) = -\frac{\varepsilon}{\sqrt{1-\bar{\alpha}_t}}$$

So predicting ε ≡ estimating the score (up to a known scalar).

Diffusion = denoising score matching across noise levels. (Song & Ermon, 2019)



The Sampling Speed Problem: T = 1000 Is Too Slow

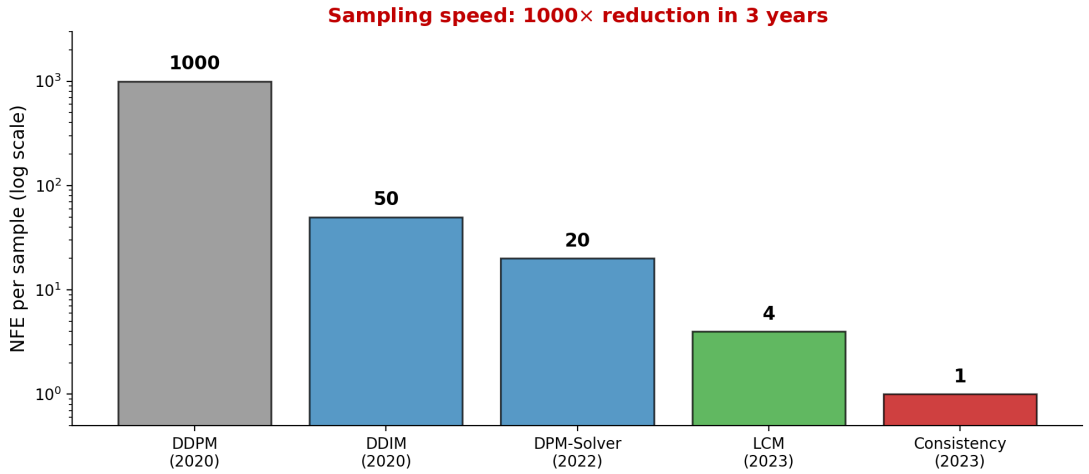
- DDPM: 1000 NFEs (Number of Function Evaluations) per sample. On a single GPU, ~30s per 256^2 image.

Three families of fixes:

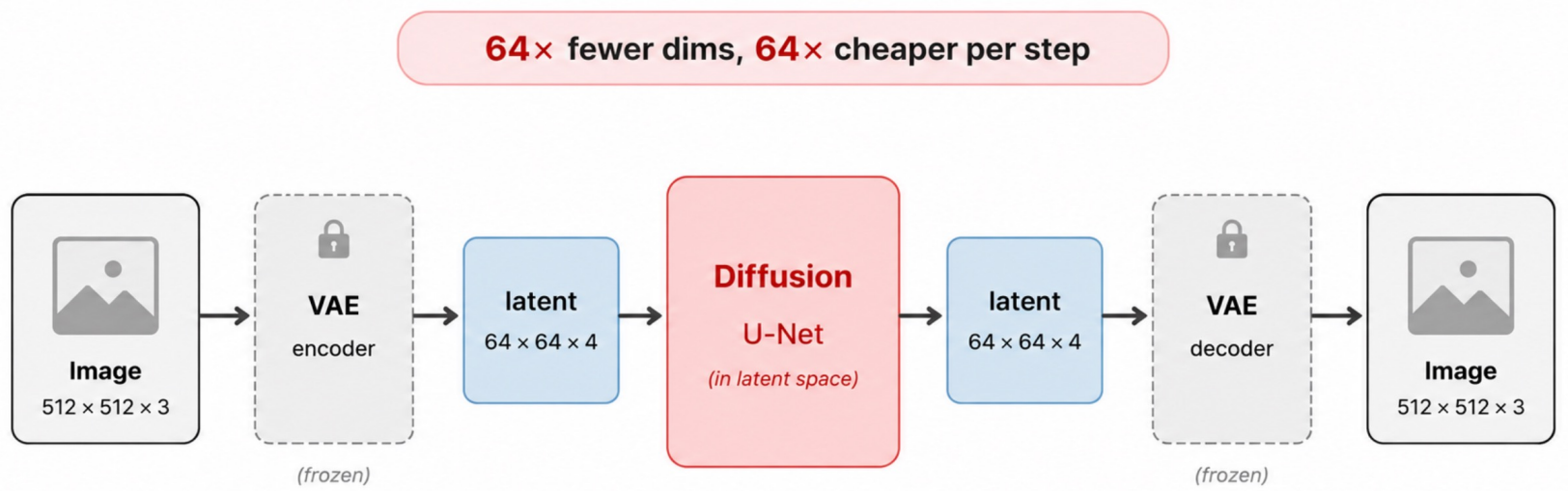
1. Faster samplers — DDIM (Song 2020), DPM-Solver (Lu 2022). 20–50 NFEs.

2. Distillation — train a fast student to match the slow teacher. 1–4 NFEs.

3. Latent-space diffusion — diffuse in a compressed latent, not pixels. Fewer dims per step.



Latent Diffusion: Why Diffuse in Pixel Space at All?



- Pixel-space diffusion at 512^2 = 786,432 dims. Painful.
- Latent Diffusion (Rombach 2022): train a VAE first to compress 512×512×3 → 64×64×4 latent. Then diffuse in latent space.
- 64× fewer dimensions → 64× cheaper per step.
- The decoder un-compresses the final latent back to pixels.

This is what made Stable Diffusion possible on consumer GPUs.

Did We Get All Three?

Model	Sample quality	Likelihood	Stable training	Fast sampling
Autoregressive	✓	✓	✓	✗
VAE	✗ blurry	~ ELBO	✓	✓
GAN	✓	✗	✗	✓
Diffusion	✓	**~ ELBO**	✓	**✗ → ~ with samplers**

Diffusion is the first model to get sample quality + stable training + (lower-bound) likelihood simultaneously.

Can we use diffusion models on language? => Yes, Diffusion Language Models.

Summary

- Diffusion = forward (fixed noise) + reverse (learned denoising)
- ε-prediction = score estimation
- Trained as a hierarchical VAE (the L24 connection)
- Latent-space + fast samplers for practicality