

Stable Diffusion Models

Aayush Bajaj

July 10, 2026

Contents

1	the forward (noising) process	1
2	from variational bound to epsilon-prediction	2
3	the reverse process and sampling	3
4	latent diffusion: the stable part	3
4.1	classifier-free guidance	4
5	toy diffusion in 1d	4
6	see also	5

diffusion models generate by learning to undo noise: destroy an image with a fixed gaussian corruption process, train a network to reverse one small step of the destruction, then chain the reversals from pure noise back to data (ho et al. 2020, *denoising diffusion probabilistic models*). **stable diffusion** (rombach et al. 2022, *high-resolution image synthesis with latent diffusion models*) runs this machinery not on pixels but in the latent space of an autoencoder, with a text-conditioned u-net doing the denoising. first the maths, then the architecture.

the generative modelling trick of the decade: replace an adversarial game with a supervised regression problem

1 the forward (noising) process

fix a variance schedule $\beta_1, \dots, \beta_T \in (0, 1)$ and define a markov chain that gradually gaussianises the data:

$$q(x_t | x_{t-1}) = \mathcal{N}(x_t; \sqrt{1 - \beta_t} x_{t-1}, \beta_t I). \quad (1)$$

the $\sqrt{1 - \beta_t}$ shrinkage is chosen so variance is *preserved*: if $\text{Var}(x_{t-1}) = I$ then $\text{Var}(x_t) = (1 - \beta_t)I + \beta_t I = I$. because compositions of these gaussians are gaussian, the chain telescopes into a **closed form** — with $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{s=1}^t \alpha_s$,

$$q(x_t | x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t} x_0, (1 - \bar{\alpha}_t)I) \iff x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} \varepsilon, \quad \varepsilon \sim \mathcal{N}(0, I). \quad (2)$$

this is the engine of cheap training: any noise level t is reachable from x_0 in one step, no simulation of the chain required. as $t \rightarrow T$ (with $\bar{\alpha}_T \approx 0$), x_T is indistinguishable from $\mathcal{N}(0, I)$ — the data has been fully forgotten.

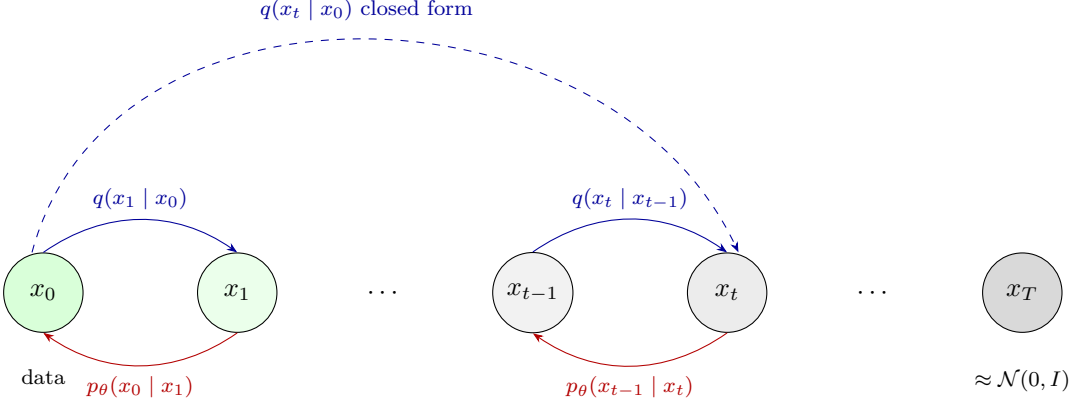


Figure 1: the diffusion markov chain: the fixed forward kernel q destroys structure left to right; the learned reverse kernel p rebuilds it right to left.

2 from variational bound to epsilon-prediction

the model is a latent-variable model $p_\theta(x_0) = \int p(x_T) \prod_{t=1}^T p_\theta(x_{t-1} | x_t) dx_{1:T}$ with gaussian reverse kernels, trained by the usual variational bound on negative log-likelihood (Goodfellow, Ian, 2016):

$$\mathbb{E}[-\log p_\theta(x_0)] \leq \mathbb{E}_q \left[\underbrace{\text{KL}(q(x_T | x_0) \| p(x_T))}_{L_T \approx 0} + \sum_{t>1} \underbrace{\text{KL}(q(x_{t-1} | x_t, x_0) \| p_\theta(x_{t-1} | x_t))}_{L_{t-1}} - \underbrace{\log p_\theta(x_0 | x_1)}_{L_0} \right]. \quad (3)$$

the pivotal fact: the forward posterior $q(x_{t-1} | x_t, x_0)$ is itself gaussian with known mean

$$\tilde{\mu}_t(x_t, x_0) = \frac{\sqrt{\bar{\alpha}_{t-1}} \beta_t}{1 - \bar{\alpha}_t} x_0 + \frac{\sqrt{\bar{\alpha}_t} (1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t} x_t, \quad (4)$$

so each L_{t-1} is a kl between gaussians — a weighted squared distance between means. now reparameterise: substitute $x_0 = (x_t - \sqrt{1 - \bar{\alpha}_t} \varepsilon) / \sqrt{\bar{\alpha}_t}$ and let the network predict the *noise* $\varepsilon_\theta(x_t, t)$ rather than the mean. the bound becomes a weighted sum of terms $\|\varepsilon - \varepsilon_\theta(x_t, t)\|^2$. the ddpm result, stated honestly: ho et al. **drop the weights** and train on the uniform-weight surrogate

$$L_{\text{simple}} = \mathbb{E}_{t \sim \mathcal{U}\{1..T\}, x_0, \varepsilon} \left[\left\| \varepsilon - \varepsilon_\theta(\sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} \varepsilon, t) \right\|^2 \right], \quad (5)$$

which is *not* the elbo but a reweighting of it that downweights the easy low- t terms — and produces better samples than the exact bound. the entire training loop is: pick an image, pick a timestep, add closed-form noise, regress the noise. no adversary, no equilibrium, compare **gans**.

3 the reverse process and sampling

the learned reverse kernel is $p_\theta(x_{t-1} | x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \sigma_t^2 I)$ with the mean expressed through the predicted noise:

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{\beta_t}{\sqrt{1 - \alpha_t}} \varepsilon_\theta(x_t, t) \right), \quad \sigma_t^2 = \beta_t \text{ (or the posterior variance; both work).} \quad (6)$$

ancestral sampling is then a loop:

- **init** $x_T \sim \mathcal{N}(0, I)$.
- **loop** for $t = T, \dots, 1$: predict $\varepsilon_\theta(x_t, t)$, form μ_θ , set $x_{t-1} = \mu_\theta + \sigma_t z$ with $z \sim \mathcal{N}(0, I)$ (no noise at $t = 1$).
- **return** x_0 .

that is T network evaluations per sample — ddpm used $T = 1000$, which is the method’s honest weakness. **ddim** (song et al. 2020, *denoising diffusion implicit models*) reinterprets the same trained ε_θ inside a non-markovian family whose deterministic member permits taking large strides over the schedule — 20–50 steps for near-identical quality, and, being deterministic, an invertible map between noise and image.

*same weights, different sampler:
ddim is an inference-time
choice, not a new model*

4 latent diffusion: the stable part

pixel-space diffusion at 512px spends most of its capacity on imperceptible high-frequency detail. latent diffusion moves the chain into a compressed space:

- **vae latent space.** a perceptually-trained autoencoder (kl-regularised, with a patch-adversarial loss on the decoder) maps a $512 \times 512 \times 3$ image to a $64 \times 64 \times 4$ latent — spatial factor 8, $\sim 48\times$ fewer numbers. diffusion happens entirely on these latents; see *autoencoders*.
- **u-net denoiser.** ε_θ is a u-net — downsampling and upsampling convolutional stages with skip connections — with the timestep injected as an embedding into every residual block.
- **cross-attention conditioning.** the prompt goes through a frozen clip text encoder into a sequence of embeddings; at several u-net resolutions, *attention* layers use image features as queries and text embeddings as keys and values, so every spatial location can consult every word.
- **decode.** after the reverse loop finishes, the vae decoder maps the final latent back to pixels — one pass, at the very end.

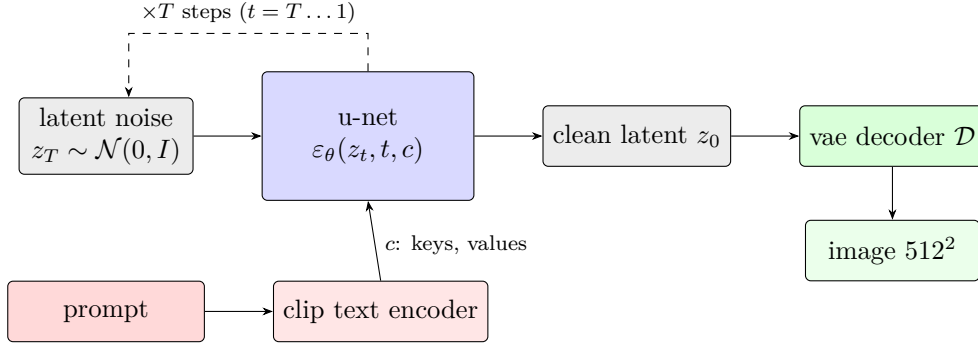


Figure 2: stable diffusion at inference: the reverse chain runs entirely in the vae’s latent space, steered by cross-attention to the text embedding; the decoder renders pixels once at the end.

4.1 classifier-free guidance

conditioning alone produces washed-out adherence to the prompt. **cfg** (ho and salimans 2022, *classifier-free diffusion guidance*) trains one network for both conditional and unconditional prediction by randomly dropping the conditioning during training, then at sampling extrapolates *past* the conditional prediction:

$$\tilde{\varepsilon}_\theta(z_t, c) = \varepsilon_\theta(z_t, \emptyset) + w(\varepsilon_\theta(z_t, c) - \varepsilon_\theta(z_t, \emptyset)), \quad (7)$$

with guidance scale $w > 1$ (stable diffusion defaults to $w \approx 7.5$). the difference vector points from “any image” toward “images matching c ”; scaling it trades diversity for prompt fidelity, and overdoing it gives the over-saturated, over-contrasty look every heavy user recognises. cost: two u-net evaluations per step.

5 toy diffusion in 1d

the full pipeline — closed-form forward noising, an mlp ε -predictor with hand-written backprop, ancestral sampling — on the most brutal test of mode coverage: a dataset of exactly two points, $x_0 \in \{-1, +1\}$. a mode-collapsing model would pick one.

```

1  import numpy as np
2
3  rng = np.random.default_rng(0)
4
5  # --- schedule: T steps, linear betas, closed-form alpha-bars
6  T = 50
7  betas = np.linspace(1e-4, 0.25, T)
8  alphas = 1 - betas
9  abar = np.cumprod(alphas)           # \bar{\alpha}_t
10
11 # --- data: two points, x0 = -1 or +1 with equal probability
12 sample_data = lambda n: rng.choice([-1.0, 1.0], size=(n, 1))
13
14 # --- epsilon-predictor: tiny mlp, input (x_t, t/T), output eps_hat
15 p = {"W1": rng.normal(0, 0.5, (2, 32)), "b1": np.zeros(32),
16      "W2": rng.normal(0, 0.5, (32, 1)), "b2": np.zeros(1)}
17 m = {k: 0.0 for k in p}; v = {k: 0.0 for k in p}
18
19 def forward(inp):
20     h = np.tanh(inp @ p["W1"] + p["b1"])
21     return h, h @ p["W2"] + p["b2"]

```

```

22
23 def adam_step(g, t, lr=2e-3):
24     for k in p:
25         m[k] = 0.9*m[k] + 0.1*g[k]
26         v[k] = 0.999*v[k] + 0.001*g[k]**2
27         p[k] -= lr * (m[k]/(1-0.9**t)) / (np.sqrt(v[k]/(1-0.999**t)) + 1e-8)
28
29 # --- training: sample t, noise with the closed form, regress the noise
30 B = 256
31 for step in range(1, 8001):
32     x0 = sample_data(B)
33     t = rng.integers(0, T, B)
34     eps = rng.normal(size=(B, 1))
35     xt = np.sqrt(abar[t])[:, None]*x0 + np.sqrt(1 - abar[t])[:, None]*eps
36     inp = np.hstack([xt, (t[:, None] + 1)/T])
37     h, eps_hat = forward(inp)
38     d = 2*(eps_hat - eps)/B # d(mse)/d(eps_hat)
39     g = {"W2": h.T @ d, "b2": d.sum(0)}
40     dh = (d @ p["W2"].T) * (1 - h**2)
41     g["W1"], g["b1"] = inp.T @ dh, dh.sum(0)
42     adam_step(g, step)
43     if step in (1, 1000, 4000, 8000):
44         print(f"step {step:>5} mse {((eps_hat - eps)**2).mean():.4f}")
45
46 # --- ancestral sampling: run the learned reverse chain from pure noise
47 n = 4000
48 x = rng.normal(size=(n, 1)) # x_T ~ N(0, 1)
49 for t in reversed(range(T)):
50     _, eps_hat = forward(np.hstack([x, np.full((n, 1), (t + 1)/T)]))
51     x = (x - betas[t]/np.sqrt(1 - abar[t]) * eps_hat) / np.sqrt(alphas[t])
52     if t > 0:
53         x += np.sqrt(betas[t]) * rng.normal(size=(n, 1))
54
55 near_pos = (np.abs(x - 1) < 0.3).mean()
56 near_neg = (np.abs(x + 1) < 0.3).mean()
57 print(f"\nsamples: mean {x.mean():+.3f}, std {x.std():.3f}")
58 print(f"fraction within 0.3 of +1: {near_pos:.3f}, of -1: {near_neg:.3f}")
59 print("mean of |x| (target 1.0):", f"{np.abs(x).mean():.3f}")

```

```

step      1 mse 2.8974
step 1000  mse 0.3682
step 4000  mse 0.2710
step 8000  mse 0.1701

```

```

samples: mean +0.051, std 1.005
fraction within 0.3 of +1: 0.473, of -1: 0.435
mean of |x| (target 1.0): 0.989

```

starting from a single gaussian blob, the reverse chain splits the mass onto *both* modes — 47% lands on +1, 44% on -1, mean of $|x|$ is 0.989 against a target of 1.0. the residual mse never reaches zero and should not: at high t the noise is genuinely unrecoverable, so a fraction of $\|\varepsilon\|^2$ is irreducible.

6 see also

- [gans](#) — the adversarial predecessor, and what mode collapse looks like
- [autoencoders](#) — the compression stage that makes latent diffusion cheap
- [transformers](#) — the cross-attention mechanism doing the text conditioning

- [cnn from scratch](#) — the convolutional anatomy of the u-net

references

Goodfellow, Ian (2016). *Deep Learning*, MIT Press.