# Diffusion Model: A Detailed Tutorial

May, 2025

## Contents

## 1 SDE Representation of the Forward Process in DDPM

During the noise-adding stage of a **DDPM**, every time-step applies the following discrete Markov chain:

$$x_i \;=\; \sqrt{1-\beta_i}\,x_{i-1} \;+\; \sqrt{\beta_i}\,\varepsilon_{i-1}, \qquad i=1,\dots,N. \tag{1}$$

To turn this into a continuous process, we let the discrete interval $\Delta t$ shrink to zero-equivalently, we consider the limit $N \to \infty$ of the Markov chain.

Before taking the limit, introduce an auxiliary noise scale $\{\tilde\beta_i = N\,\beta_i\}_{i=1}^{N}$ and rewrite

$$x_i \;=\; \sqrt{1-\frac{\tilde\beta_i}{N}}\,x_{i-1} \;+\; \sqrt{\frac{\tilde\beta_i}{N}}\,\varepsilon_{i-1}, \qquad i=1,\dots,N. \tag{2}$$

As $N \to \infty$, the sequence $\{\tilde\beta_i\}_{i=1}^{N}$ becomes a continuous schedule $\beta(t)$ on $t \in [0,1]$. Set

$$\Delta t = \frac{1}{N}.$$

At each $\Delta t$, the continuous functions $\beta(t)$, $x(t)$, $\varepsilon(t)$ coincide with their discrete counterparts:

$$\beta\big(\tfrac{i}{N}\big) = \tilde\beta_i, \qquad x\big(\tfrac{i}{N}\big) = x_i, \qquad \varepsilon\big(\tfrac{i}{N}\big) = \varepsilon_i.$$

For $t \in \{0, \frac{1}{N}, \dots, \frac{N-1}{N}\}$ and $t+\Delta t = t + \frac{1}{N}$, we rewrite the update with continuous notation:

$$\begin{aligned} x(t+\Delta t) &= \sqrt{1-\beta(t+\Delta t)\,\Delta t}\; x(t) + \sqrt{\beta(t+\Delta t)\,\Delta t}\;\varepsilon(t) && (3)\\ &\approx x(t) - \tfrac{1}{2}\beta(t+\Delta t)\,\Delta t\,x(t) + \sqrt{\beta(t+\Delta t)\,\Delta t}\;\varepsilon(t) && (4)\\ &\approx x(t) - \tfrac{1}{2}\beta(t)\,\Delta t\,x(t) + \sqrt{\beta(t)\,\Delta t}\;\varepsilon(t). && (5) \end{aligned}$$

(The second line holds when $\Delta t \ll 1$.) Thus

$$x(t+\Delta t) - x(t) \;\approx\; -\tfrac{1}{2}\beta(t)\,\Delta t\,x(t) + \sqrt{\beta(t)\,\Delta t}\;\varepsilon(t).$$

Letting $\Delta t \to 0$ gives the stochastic differential equation (SDE)

$$\boxed{\mathrm{d}x = -\tfrac{1}{2}\beta(t)\,x\,\mathrm{d}t + \sqrt{\beta(t)}\,\mathrm{d}w}\,. \tag{6}$$

## 2 SDE Representation of the Reverse Process in DDPM

Using probabilistic language, Eq. (6) can be generalized to the stochastic differential equation (SDE)

$$\mathrm{d}x \;=\; f_t(x)\,\mathrm{d}t \;+\; g_t\,\mathrm{d}w, \tag{7}$$

which likewise describes the forward noise-adding Markov chain. Whether the chain is *linear* depends on whether $f_t(x)$ is linear in $x$. The corresponding one-step conditional density is

$$p\big(x_{t+\Delta t} \mid x_t\big) \;=\; \mathcal{N}\Big(x_{t+\Delta t};\, x_t + f_t(x_t)\,\Delta t,\; g_t^2\,\Delta t\, I\Big) \;\propto\; \exp\Big(-\frac{\|x_{t+\Delta t} - x_t - f_t(x_t)\,\Delta t\|^2}{2\,g_t^2\,\Delta t}\Big). \tag{8}$$

(The constant normalization factor is omitted.)

Following the DDPM philosophy—"learning to build by watching demolition"— we ultimately wish to find the reverse density $p(x_t \mid x_{t+\Delta t})$. Applying Bayes' rule,

$$\begin{aligned} p\big(x_t \mid x_{t+\Delta t}\big) &= \frac{p(x_{t+\Delta t} \mid x_t)\,p(x_t)}{p(x_{t+\Delta t})}\\ &\propto \exp\Big(-\frac{\|x_{t+\Delta t} - x_t - f_t(x_t)\,\Delta t\|^2}{2\,g_t^2\,\Delta t} + \log p(x_t) - \log p(x_{t+\Delta t})\Big). \end{aligned} \tag{9}$$

Because $\Delta t$ is small, $p(x_{t+\Delta t} \mid x_t)$ is appreciable only when $x_{t+\Delta t}$ is close to $x_t$; the same is true for the reverse density. Hence we expand

$$\log p(x_{t+\Delta t}) \;\approx\; \log p(x_t) + (x_{t+\Delta t} - x_t)^\top \nabla_{x_t} \log p(x_t) + \Delta t\, \partial_t \log p(x_t).$$

Substituting and collecting terms,

$$p\big(x_t \mid x_{t+\Delta t}\big) \;\propto\; \exp\Big(-\frac{\big\|x_{t+\Delta t} - x_t - \big[f_t(x_t) - g_t^2 \nabla_{x_t} \log p(x_t)\big]\Delta t\big\|^2}{2\, g_t^2\, \Delta t} + \mathcal{O}(\Delta t)\Big). \tag{10}$$

Letting $\Delta t \to 0$ gives

$$p\big(x_t \mid x_{t+\Delta t}\big) \;\propto\; \exp\Big(-\frac{\big\|x_{t+\Delta t} - x_t - \big[f_{t+\Delta t}(x_{t+\Delta t}) - g_{t+\Delta t}^2 \nabla_{x_{t+\Delta t}} \log p(x_{t+\Delta t})\big]\Delta t\big\|^2}{2\, g_{t+\Delta t}^2\, \Delta t}\Big). \tag{11}$$

Hence $p(x_t \mid x_{t+\Delta t})$ is approximately Gaussian with mean

$$x_{t+\Delta t} - \Big[f_{t+\Delta t}(x_{t+\Delta t}) - g_{t+\Delta t}^2 \nabla_{x_{t+\Delta t}} \log p(x_{t+\Delta t})\Big]\Delta t,$$

and covariance $g_{t+\Delta t}^2\, \Delta t\, I$. Taking $\Delta t \to 0$ recovers the reverse-time SDE

$$\mathrm{d}x \;=\; \big[f_t(x) - g_t^2 \nabla_x \log p(x)\big]\, \mathrm{d}t + g_t\, \mathrm{d}w. \tag{12}$$

For the linear forward Markov chain in Eq. (1), $f_t(x) = -\frac{1}{2}\beta(t)x$ and $g_t = \sqrt{\beta(t)}$, yielding

$$\boxed{\mathrm{d}x = \big[-\tfrac{1}{2}\beta(t)x - \beta(t)\nabla_x \log p(x)\big]\, \mathrm{d}t + \sqrt{\beta(t)}\, \mathrm{d}w}\,. \tag{13}$$

## 3 Incorporating a Neural Network

Given the reverse-time SDE in Eq. (13), the noise schedule $\beta(t)$ is known at every step; to complete the *generation* (or "building-up") process we need only the score $\nabla_x \log p(x)$. Here $p(x)$ denotes the marginal distribution of $x$ at time $t$. For a *linear* forward Markov process this marginal can be written in closed form, but doing so requires an average over *all* training samples $x_0$, which is computationally expensive and does not generalize well. We therefore train a neural network to *directly* approximate $\nabla_x \log p(x)$.

The diffusion process specifies the forward transition density $p(x_{t+\Delta t} \mid x_t)$. Integrating these infinitesimal transitions in sequence gives

$$p(x_t \mid x_0) \;=\; \lim_{\Delta t \to 0} \int \cdots \int p(x_t \mid x_{t-\Delta t})\, p(x_{t-\Delta t} \mid x_{t-2\Delta t}) \cdots p(x_{\Delta t} \mid x_0)\, dx_{t-\Delta t} \cdots dx_{\Delta t}. \tag{14}$$

When the forward chain is linear, the expression above admits a closed-form solution (not guaranteed if the chain is nonlinear). Consequently the marginal at time $t$ is

$$p(x_t) \;=\; \int p(x_t \mid x_0)\, p(x_0)\, dx_0 \;=\; \mathbb{E}_{x_0}\big[p(x_t \mid x_0)\big], \tag{15}$$

and

$$\nabla_{x_t} \log p(x_t) \;=\; \frac{\mathbb{E}_{x_0}\big[p(x_t \mid x_0)\, \nabla_{x_t} \log p(x_t \mid x_0)\big]}{\mathbb{E}_{x_0}\big[p(x_t \mid x_0)\big]}. \tag{16}$$

**Lemma 1** (Weighted MSE minimizer). *Let $x \in \mathbb{R}^d$ be a random vector with $\mathbb{E}\|x\|^2 < \infty$, and let $w \geq 0$ be a non-negative random weight on the same space with $\mathbb{E}[w] > 0$. For any fixed vector $y \in \mathbb{R}^d$, define the weighted mean-squared error*

$$J(y) \;=\; \mathbb{E}\big[w\, \|y - x\|^2\big].$$

*Then $J(y)$ is strictly convex in $y$ and attains its unique minimizer at*

$$y^\star \;=\; \frac{\mathbb{E}[w\, x]}{\mathbb{E}[w]}.$$

*When $w \equiv 1$, this reduces to the classical result $y^\star = \mathbb{E}[x]$.*

Choose

$$w = p(x_t \mid x_0), \quad x = \nabla_{x_t} \log p(x_t \mid x_0), \quad y = s_\theta(x_t, t),$$

and define the loss

$$\mathcal{L}(s_\theta) \;=\; \mathbb{E}_{x_0}\big[p(x_t \mid x_0)\,\big\|s_\theta(x_t,t) - \nabla_{x_t} \log p(x_t \mid x_0)\big\|^2\big]. \tag{17}$$

By the lemma, this loss is minimized at

$$s_\theta^\star(x_t,t) \;=\; \frac{\mathbb{E}_{x_0}\big[p(x_t \mid x_0)\,\nabla_{x_t} \log p(x_t \mid x_0)\big]}{\mathbb{E}_{x_0}\big[p(x_t \mid x_0)\big]} \;=\; \nabla_{x_t} \log p(x_t). \tag{18}$$

Since the denominator merely rescales the loss, we drop it for simplicity. Expanding the outer expectation over $x_0$ yields

$$\mathcal{L}(s_\theta) \;=\; \int \mathbb{E}_{x_0|x_t}\big[\|s_\theta(x_t,t) - \nabla_{x_t} \log p(x_t \mid x_0)\|^2\big] p(x_t)\,dx_t \;=\; \mathbb{E}_{x_0,\,x_t\sim p(x_t|x_0)\,\tilde{p}(x_0)}\Big[\|\,s_\theta(x_t,t) - \nabla_{x_t} \log p(x_t \mid x_0)\|^2\Big] \tag{19}$$

**Forward-process reparameterization.** From the DDPM forward dynamics one can write $x_t$ in closed form:

$$x_t = \alpha_t x_{t-1} + \beta_t \varepsilon_t = \cdots = (\alpha_t \cdots \alpha_1)\,x_0 + \textstyle\sum_{k=1}^{t} \big(\alpha_t \cdots \alpha_{k+1}\big)\beta_k \varepsilon_k.$$

Because $(\alpha_t \cdots \alpha_1)^2 + (\alpha_t \cdots \alpha_2)^2 \beta_1^2 + \cdots + \beta_t^2 = 1$, we may write

$$x_t = \tilde{\alpha}_t x_0 + \tilde{\beta}_t \varepsilon, \qquad \tilde{\alpha}_t^2 + \tilde{\beta}_t^2 = 1,$$

with $p(x_t \mid x_0) = \mathcal{N}\big(x_t;\, \tilde{\alpha}_t x_0, \tilde{\beta}_t^2 I\big)$.

Hence

$$\nabla_{x_t} \log p(x_t \mid x_0) = -\frac{1}{\tilde{\beta}_t^2}\big(x_t - \tilde{\alpha}_t x_0\big) = -\frac{1}{\tilde{\beta}_t}\,\varepsilon.$$

Let $s_\theta(x_t,t) = -\varepsilon_\theta(x_t,t)/\bar{\beta}_t$; dropping the constant factor $1/\bar{\beta}_t^2$ gives the final training loss

$$\boxed{\mathcal{L}(\varepsilon_\theta) = \mathbb{E}_{x_0\sim\tilde{p}(x_0),\,\varepsilon\sim\mathcal{N}(0,I)}\Big[\big\|\varepsilon_\theta\big(\bar{\alpha}_t x_0 + \bar{\beta}_t \varepsilon,\ t\big) - \varepsilon\big\|^2\Big]}. \tag{20}$$

Minimizing this loss forces the network $\varepsilon_\theta$ to approximate the true score $\nabla_{x_t} \log p(x_t)$ at *every* time step $t$.

## 4 Discretizing the Reverse Process: A Stepwise View

Sections 1 and 2 showed that a trained DDPM can be interpreted as a stochastic process that models the *reverse* (denoising) dynamics. Here we discretize those continuous dynamics and make explicit how to sample $x_{t-1}$ from a given $x_t$.

By Bayes' theorem

$$p(x_{t-1} \mid x_t) \;=\; \frac{p(x_t \mid x_{t-1})\,p(x_{t-1})}{p(x_t)}. \tag{21}$$

Because the marginals $p(x_{t-1})$ and $p(x_t)$ are unknown in closed form, this expression is not directly usable (the continuous-time derivation in Section 2 sidestepped them by taking $\Delta t \to 0$).

Instead, condition on the (unknown) clean image $x_0$:

$$p(x_{t-1} \mid x_t, x_0) \;=\; \frac{p(x_t \mid x_{t-1}, x_0)\,p(x_{t-1} \mid x_0)}{p(x_t \mid x_0)}. \tag{22}$$

Now every factor is known analytically, yielding

$$p(x_{t-1} \mid x_t, x_0) \;=\; \mathcal{N}\Big(x_{t-1};\ \frac{\alpha_t \bar{\beta}_{t-1}^2}{\bar{\beta}_t^2}\,x_t + \frac{\bar{\alpha}_{t-1}\beta_t^2}{\bar{\beta}_t^2}\,x_0,\ \frac{\bar{\beta}_{t-1}^2}{\bar{\beta}_t^2}\,I\Big). \tag{23}$$

Because $x_0$ is unavailable during inference, we replace it by a learned predictor $\hat{\mu}(x_t)$. If $\hat{\mu}(x_t)$ is trained with the loss $\|x_0 - \hat{\mu}(x_t)\|^2$, then

$$p(x_{t-1} \mid x_t) \;\approx\; p\Big(x_{t-1} \mid x_t, x_0 = \hat{\mu}(x_t)\Big) = \mathcal{N}\Big(x_{t-1}; \tfrac{\alpha_t \bar{\beta}_{t-1}^2}{\bar{\beta}_t^2}\, x_t + \tfrac{\bar{\alpha}_{t-1}\beta_t^2}{\bar{\beta}_t^2}\, \hat{\mu}(x_t),\; \tfrac{\bar{\beta}_{t-1}^2}{\bar{\beta}_t^2}\, I\Big).$$

Because $x_t = \bar{\alpha}_t x_0 + \bar{\beta}_t \varepsilon$, one may solve for $x_0$ as $x_0 = \frac{1}{\bar{\alpha}_t}(x_t - \bar{\beta}_t \varepsilon)$. This motivates

$$\hat{\mu}(x_t) = \frac{1}{\bar{\alpha}_t}\Big(x_t - \bar{\beta}_t\, \varepsilon_\theta(x_t, t)\Big),$$

where $\varepsilon_\theta$ is the usual noise-predicting UNet.

Training $\varepsilon_\theta$ with

$$\|x_0 - \hat{\mu}(x_t)\|^2 = \frac{\bar{\beta}_t^2}{\bar{\alpha}_t^2}\Big\|\varepsilon - \varepsilon_\theta(\bar{\alpha}_t x_0 + \bar{\beta}_t \varepsilon,\, t)\Big\|^2$$

reduces (up to a scalar) to the simplified DDPM loss.

Substituting $\hat{\mu}(x_t)$ back yields

$$p(x_{t-1} \mid x_t) \;\approx\; \mathcal{N}\Big(x_{t-1}; \frac{1}{\alpha_t}\Big(x_t - \tfrac{\beta_t^2}{\bar{\beta}_t}\varepsilon_\theta(x_t, t)\Big),\; \frac{\bar{\beta}_{t-1}^2 \beta_t^2}{\bar{\beta}_t^2}\, I\Big),$$

so that a single sampling step is

$$x_{t-1} = \frac{1}{\alpha_t}\Big(x_t - \tfrac{\beta_t^2}{\bar{\beta}_t}\, \varepsilon_\theta(x_t, t)\Big) + \frac{\bar{\beta}_{t-1}\beta_t}{\bar{\beta}_t}\, \varepsilon, \qquad \varepsilon \sim \mathcal{N}(0, I). \tag{24}$$

## 5 Interpreting Diffusion Models via the Score Function

The *score function* of a density is defined as its log-likelihood gradient, $s(x) = \nabla_x \log p(x)$, pointing in the direction of steepest increase in probability. In Langevin dynamics and diffusion models, this score acts as the drift term that "pulls" noisy samples back toward high-density regions, as schematically illustrated below.

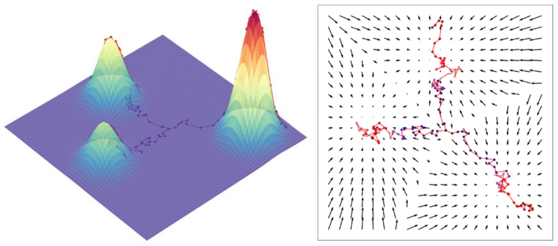

Figure 1: Intuition: score-based drift guides noisy particles toward the data manifold.

Accordingly, the reverse process trains the network to approximate the score of each noisy image $x_t$, effectively teaching it how to guide particles from $p(x_{t+1})$ back to $p(x_t)$.

## 6 The DPS Algorithm: Posterior-Consistent Generation

### 6.1 Algorithmic Rationale

Armed with the preceding background, we now examine the **DPS (Denoising Posterior Sampling)** algorithm in a practical setting.

In an *unconditional* diffusion model the only guidance is the prior score $\nabla_x \log p(x)$; the learned "force field" merely pulls noisy samples toward the natural-image manifold. In real applications—say, image denoising—we need the final output to be both *realistic* (prior fidelity) and *consistent* with a noisy observation $y$ (data fidelity). Hence we upgrade the marginal $p(x)$ at every step to a conditional $p(x \mid y)$. In the reverse SDE, the network must now predict $\nabla_x \log p(x \mid y)$ instead of $\nabla_x \log p(x)$.

Bayes' rule gives

$$\nabla_x \log p(x \mid y) = \nabla_x \log p(x) + \nabla_x \log p(y \mid x).$$

The prior term can be pretrained as usual, but $\nabla_x \log p(y \mid x)$ lacks a closed form because the likelihood is time-dependent and we only know the relation between $y$ and the clean image $x_0$.

Assume the observation model $y = A(x_0) + n$, with $A$ an imaging operator and $n$ additive noise. Because the likelihood $p(y \mid x_t)$ has no analytic form, we relate it to the known $p(y \mid x_0)$ via conditional independence:

$$p(y \mid x_t) = \int p(y \mid x_0, x_t)\, p(x_0 \mid x_t)\, dx_0 = \int p(y \mid x_0)\, p(x_0 \mid x_t)\, dx_0 = \mathbb{E}_{x_0 \sim p(x_0 \mid x_t)}[p(y \mid x_0)].$$

Applying Jensen's inequality,

$$p(y \mid x_t) \approx p\big(y \mid \hat{x}_0\big), \qquad \hat{x}_0 := \mathbb{E}[x_0 \mid x_t].$$

Because the forward process satisfies $x_t = \sqrt{\bar{\alpha}}\, x_0 + \sqrt{1-\bar{\alpha}}\, z$, one finds

$$\hat{x}_0(x_t) = \frac{1}{\sqrt{\bar{\alpha}}}\Big(x_t + (1-\bar{\alpha})\, \nabla_{x_t} \log p_t(x_t)\Big).$$

Assuming Gaussian observation noise

$$p(y \mid x_0) \propto \exp\Big(-\tfrac{1}{2\sigma^2} \|y - A(x_0)\|_2^2\Big),$$

we obtain

$$\nabla_{x_t} \log p_t(y \mid x_t) \approx -\frac{1}{\sigma^2}\, \nabla_{x_t} \|y - A\big(\hat{x}_0(x_t)\big)\|_2^2.$$

Hence the *posterior* score is approximated by

$$\nabla_{x_t} \log p_t(x_t \mid y) \approx s_\theta^*(x_t, t) - \rho\, \nabla_{x_t} \|y - A(\hat{x}_0)\|_2^2, \qquad \rho := \frac{1}{\sigma^2}.$$

## 6.2 Practical Implementation

1. **Initialization.** Start the reverse chain with pure noise $x_N \sim \mathcal{N}(0, I)$.
2. **Backward iteration.** For $i = N-1, \dots, 0$:
   (i) Compute the prior score with the pretrained model $\hat{s} = s_\theta(x_i, i) \approx \nabla_{x_i} \log p_t(x_i)$.
   (ii) **Tweedie update** (estimate a clean image)
   $$\hat{x}_0 = \frac{1}{\sqrt{\bar{\alpha}_i}}\big(x_i + (1-\bar{\alpha}_i)\hat{s}\big).$$
   (iii) **Prior-only DDPM step**
   $$x'_{i-1} = \sqrt{\frac{\bar{\alpha}_{i-1}(1-\alpha_i)}{1-\bar{\alpha}_i}}\, x_i + \sqrt{\frac{\bar{\alpha}_{i-1}\beta_i}{1-\bar{\alpha}_i}}\, \hat{x}_0 + \tilde{\sigma}_i\, z, \;\; z \sim \mathcal{N}(0, I).$$
   (iv) **Likelihood correction** (Gaussian case)
   $$x_{i-1} = x'_{i-1} - \zeta_i\, A^\top\big(A(\hat{x}_0) - y\big),$$
   where $\zeta_i$ trades off prior versus data fidelity.
3. **Output.** After the loop, return the reconstruction $\hat{x}_0$—now consistent with both the prior (realism) and the observation $y$ (data fidelity).

## 7 DDIM: A Higher-Level Perspective on DDPM

Why model diffusion with the *solution* of an SDE—i.e. a *stochastic* process? Because an SDE captures the key feature of DDPM denoising: given the previous state, the next state is *not* deterministic; only its *distribution* is specified. Suppose all images possessing a certain attribute follow a latent distribution $q$. We want a model that, when fed a noisy image $x_t$ at noise level $t$, predicts *the distribution of* $x_{t-1}$, not a single value. By sampling from that distribution at each step we eventually obtain a random draw from $q$. If a large noise level $T$ converts any image to roughly standard normal, then by $T$ steps of prediction and sampling we can transform pure Gaussian noise into a valid sample from $q$.

Once the forward relation $x_t \leftrightarrow x_0$ is available in closed form, the step-by-step noising is unnecessary; the time parameter $t$ merely controls the noise intensity. This observation leads to DDIM: because the *result* does not depend on $p(x_t \mid x_{t-1})$, we can drop the "build-and-demolish" construction entirely.

In principle, even without an explicit $p(x_t \mid x_{t-1})$, the conditional $p(x_{t-1} \mid x_t, x_0)$ is solvable; indeed, the solution set is larger and easier to characterize. All that is required is the *marginal-consistency* condition

$$\int p(x_{t-1} \mid x_t, x_0)\, p(x_t \mid x_0)\, dx_t \;=\; p(x_{t-1} \mid x_0).$$

With undetermined coefficients we can solve this directly. More generally, assume

$$p(x_{t-1} \mid x_t, x_0) \;=\; \mathcal{N}\big(x_{t-1};\, \kappa_t x_t + \lambda_t x_0,\, \sigma_t^2 I\big),$$

with $\kappa_t, \lambda_t, \sigma_t$ to be determined. Using $p(x_{t-1} \mid x_0)$ and $p(x_t \mid x_0)$ one obtains a *family* of solutions parameterized by the free variance $\sigma_t$. Training is unaffected (the saved model is unchanged), but generation now has a tunable parameter $\sigma_t$—the key novelty introduced by DDIM.

In the building-demolishing metaphor, we know what the fully demolished building looks like $\big(p(x_t \mid x_0)$ and $p(x_{t-1} \mid x_0)\big)$, but not how each individual plank is removed $\big(p(x_t \mid x_{t-1})\big)$. If $x_t$ lets us estimate $x_0$, then teaching the model, given $(x_0, x_t)$, to recover the intermediate state *is* to learn every reverse step $p(x_{t-1} \mid x_t)$—in other words, to "rebuild" the house one floor at a time without ever specifying the original demolition plan.

## 8 Concrete Implementation of DDPM

### 8.1 Training Procedure

- Sample an image $x_0$ from the training set $\big(x_0 \sim q(x_0)\big); Drawarandomtimestept \sim \text{Uniform}(1, \ldots, T)$;
- Sample Gaussian noise $\varepsilon \sim \mathcal{N}(0, \mathbf{I})$;
- Compute the loss

  $$loss \;=\; \Big\| \varepsilon - \varepsilon_\theta\big(\sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1-\bar{\alpha}_t}\, \varepsilon,\; t\big) \Big\|^2,$$

  where $\varepsilon_\theta$ is the UNet-based denoising network;
- Back-propagate the loss and update the model parameters; repeat until convergence.

### 8.2 Inference (Sampling) Procedure

- Draw an initial latent $x_T \sim \mathcal{N}(0, \mathbf{I})$;
- For $t = T, T-1, \ldots, 1$:
  - Sample fresh noise $\varepsilon \sim \mathcal{N}(0, \mathbf{I})$;
  - Use Eq. (24) to obtain $x_{t-1}$ from $x_t$.
- The final output is $x_0'$.

## 8.3 Extension to Text-to-Image Generation

To turn a DDPM into a text-to-image diffusion model, replace each self-attention block in the UNet with a *cross-attention* block, feeding the encoded text prompt $y$ as $\mathbf{K}$ and $\mathbf{V}$ while retaining the latent features as $\mathbf{Q}$. The training loss becomes

$$\mathit{loss} \;=\; \Big\|\varepsilon - \varepsilon_\theta\big(\sqrt{\bar{\alpha}_t}\,x_0 + \sqrt{1-\bar{\alpha}_t}\,\varepsilon,\ t,\ y\big)\Big\|^2.$$

# 9 DreamFusion: From 2-D to 3-D Generation

Diffusion models have achieved impressive text-to-*image* generation, yet text-to-*3-D* remains difficult because large text–shape pairs are scarce. *DreamFusion: Text-to-3D Using 2-D Diffusion* remedies this by using a *pre-trained text-to-image* diffusion model to supervise a 3-D generator.

## 9.1 Conceptual Overview

Let $\theta$ denote the parameters of a volume renderer $g(\cdot)$ (e.g. a Gaussian NeRF). Render $x = g(\theta)$, feed it into a frozen U-Net, and define the diffusion loss

$$\mathcal{L}_{\text{Diff}}(\varphi, x = g(\theta)) = \mathbb{E}_{t,\varepsilon}\big[\|\,\hat{\varepsilon}_\varphi(\alpha_t g(\theta) + \sigma_t\varepsilon;\, y, t) - \varepsilon\|^2\big],$$

where $\hat{\varepsilon}_\varphi$ is the frozen U-Net predictor. If $g(\theta)$ produces photorealistic pixels, the U-Net predicts noise well and the loss is low; otherwise it is high.

With $z_t = \alpha_t x + \sigma_t\varepsilon$ and $x = g(\theta)$,

$$\nabla_\theta\mathcal{L}_{\text{Diff}} = \mathbb{E}_{t,\varepsilon}\big[2(\hat{\varepsilon}_\varphi(z_t) - \varepsilon)\,\partial_{z_t}\hat{\varepsilon}_\varphi(z_t)\,\partial_x z_t\,\partial_\theta x\big].$$

Here $\partial_{z_t}\hat{\varepsilon}_\varphi$ (the U-Net Jacobian) is enormous; propagating it would require a full backward pass *per* pixel sample and yields unstable gradients at high noise levels.

Therefore DreamFusion *drops* the Jacobian, retaining only the residual $\hat{\varepsilon}_\varphi(z_t) - \varepsilon$. The gradient becomes

$$\nabla_\theta\mathcal{L}_{\text{SDS}} = \mathbb{E}_{t,\varepsilon}\big[w(t)\big(\hat{\varepsilon}_\varphi(z_t; y, t) - \varepsilon\big)\,\partial_\theta x\big],$$

the celebrated **score-distillation sampling (SDS)** gradient.

Crucially, SDS is **not** a heuristic; it performs *probability-density distillation* in parameter space: updating $\theta$ with the SDS gradient equals minimizing a *weighted KL divergence* between (1) the noisy distribution of the rendered image and (2) the true diffusion distribution at the same noise level $t$.

## 9.2 Density Distillation in Parameter Space

**KL divergence.** For densities $p(z)$ and $q(z)$,

$$\mathrm{KL}(p\|q) = \int p(z)\log\frac{p(z)}{q(z)}\,dz = \mathbb{E}_{z\sim p}[\log p(z) - \log q(z)].$$

If $q$ encodes $p$, the extra code length over the optimal $H(p)$ is exactly $\mathrm{KL}(p\|q)$.

**Equivalence to SDS.** Let $q(z_t \mid x) = \mathcal{N}(z_t; \alpha_t x, \sigma_t^2 I)$ with $x = g(\theta)$, and $p_\phi(z_t \mid y)$ be the true diffusion distribution at level $t$. One finds

$$\nabla_\theta\mathrm{KL}\big(q\|p_\phi\big) = \mathbb{E}_{t,\varepsilon}\big[w(t)\big(\epsilon_\phi(z_t) - \varepsilon\big)\,\partial_\theta x\big],$$

identical to $\nabla_\theta\mathcal{L}_{\text{SDS}}$.

## 9.3 Practical Pipeline (Stable DreamFusion)

Each step:

**1. Rendering.** Render a low-resolution image $x = g(\theta)$ ($[B, 3, H, W]$), then upsample.
**2. Forward noise.** Encode the high-res image with the VAE of Stable Diffusion; sample $t \sim \text{Uniform}(1, \ldots, T)$ and add noise.
**3. Denoising.** Run the noisy latent through the frozen U-Net to obtain $\hat{\varepsilon}$.
**4. Back-prop SDS.** Because the U-Net is frozen, stop its gradient and compute

```
w        = (1 - self.alphas[t])                # weight w(t)
grad     = w[:, None, None, None] * (noise_pred - noise)
target   = (latents - grad).detach()           # stop gradient
loss_sds = 0.5 * F.mse_loss(
               latents, target, reduction="sum") / batch_size
```

**5. Update $\theta$.** Back-propagate `loss_sds` through the NeRF MLP only; iterate until convergence.

# 10 ProlificDreamer: Optimizing Parameter *Distributions*

The previous section (DreamFusion) trains a *single* NeRF parameter set $\theta$ via diffusion and the SDS loss, but such a point estimate can suffer from *low diversity*. Inspired by 2-D diffusion, where one first learns a *distribution* over images and then samples from it, we ask: can we likewise learn a *distribution* $\mu$ over 3-D parameters $\theta$ and then sample a diverse $\theta$ from $\mu$? This is the core idea behind **ProlificDreamer**.

## 10.1 Principle Overview

### 10.1.1 Variational Score Distillation (VSD)

For a text prompt $y$ there exists a distribution of all 3-D scenes consistent with that prompt. Denote the distribution of NeRF parameters by $\mu(\theta \mid y)$. Rendering with viewpoint $c$ yields $x_0 = g(\theta, c)$. Let $q_0^\mu(x_0 \mid c, y)$ be the image distribution obtained by sampling $\theta \sim \mu(\cdot \mid y)$ and rendering at $c$. Let $p_0(x_0 \mid y)$ be the distribution produced by a frozen text-to-image diffusion model conditioned on $y$.

We wish to minimize their KL divergence:

$$\min_{\mu} D_{\mathrm{KL}}\big(q_0^\mu(x_0 \mid y) \,\|\, p_0(x_0 \mid y)\big),$$

a *classical variational-inference* objective.

To ensure closeness *at every noise level*, we instead minimize the weighted KL along the entire noising trajectory:

$$\mu^* = \arg\min_{\mu} \mathbb{E}_{t,c}\Big[\tfrac{\sigma_t}{\alpha_t}\, w(t)\, D_{\mathrm{KL}}\big(q_t^\mu(x_t \mid c, y) \,\|\, p_t(x_t \mid y)\big)\Big]. \tag{25}$$

### 10.1.2 Updating $\mu$ via Particle-Based Variational Inference

Treat $n$ parameter sets $\{\theta_i\}_{i=1}^n$ as *particles* representing $\mu$ ($n = 4$ in the original paper). Gradient descent with step size $\eta \to 0$ leads to an ODE:

$$\dot{\theta}_\tau = -\nabla_\theta L(\theta_\tau),$$

known here as the *Wasserstein gradient flow of VSD*. Starting from $\theta_0 \sim \mu_0(\theta \mid y)$ and integrating the ODE yields the optimal distribution $\mu_\tau$ as $\tau \to \infty$.

Define

$$\frac{d\theta_\tau}{d\tau} = -\mathbb{E}_{t,\varepsilon,c}\left[\omega(t)\Big(-\sigma_t \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t \mid y) + \sigma_t \nabla_{\mathbf{x}_t} \log q_t^{\mu_\tau}(\mathbf{x}_t \mid c, y)\Big)\frac{\partial g(\theta_\tau, c)}{\partial \theta_\tau}\right].$$

Real-image scores use the frozen diffusion model $\epsilon_{\text{pretrain}}$; rendered-image scores use a learnable network $\epsilon_\phi$ (a lightweight UNet or LoRA-tuned clone of the pretrained network). Alternating updates yield

$$\boxed{\nabla_\theta \mathcal{L}_{\text{VSD}}(\theta) = \mathbb{E}_{t,\varepsilon,c}\Big[\omega(t)\big(\epsilon_{\text{pretrain}}(\mathbf{x}_t, t, y) - \epsilon_\phi(\mathbf{x}_t, t, c, y)\big)\frac{\partial g(\theta, c)}{\partial \theta}\Big]}.$$

SDS is the special case where $\mu$ is a Dirac delta.

## 10.2 Practical Workflow (Pseudo-Code)

**Inputs:** number of particles $n$, prompt $y$, frozen score $\epsilon_{\text{pretrain}}$, learning rates $\eta_1, \eta_2$, renderer $g(\theta, c)$, noise schedule $\{\alpha_t, \sigma_t, \omega(t)\}$.

**Initialization:** $\{\theta_0^{(i)}\}_{i=1}^n \sim \text{InitPrior}, \quad \phi_0 \leftarrow \text{InitWeights}$.

**Main Loop:** for $k = 0, \dots$ until convergence

```
1. Sample particle index and camera: $i_k \sim \text{Uniform}\{1, \dots, n\},\ c_k \sim p(c)$.
2. Render clean image: $x_0^{(k)} = g(\theta_k^{(i_k)}, c_k)$.
3. Noise it: $t_k \sim \text{U}(0,1),\ \varepsilon_k \sim \mathcal{N}(0, I),\ x_{t_k}^{(k)} = \alpha_{t_k} x_0^{(k)} + \sigma_{t_k} \varepsilon_k$.
4. Compute real and rendered scores: $\hat{\varepsilon}_k^{\text{real}} = \epsilon_{\text{pretrain}}(x_{t_k}^{(k)}, t_k, y),\ \hat{\varepsilon}_k^{\text{rend}} = \epsilon_{\phi_k}(x_{t_k}^{(k)}, t_k, c_k, y)$.
5. Update geometry:
   $\theta_{k+1}^{(i_k)} = \theta_k^{(i_k)} - \eta_1\, \omega(t_k)\big(\hat{\varepsilon}_k^{\text{real}} - \hat{\varepsilon}_k^{\text{rend}}\big)\frac{\partial g(\theta_k^{(i_k)}, c_k)}{\partial \theta^{(i_k)}}$.
6. Update $\phi$: minimize $\|\epsilon_{\phi_k}(x_{t_k}^{(k)}, t_k, c_k, y) - \varepsilon_k\|_2^2$ with step $\eta_2$.
```

Return the final particle set $\{\theta_{\text{final}}^{(i)}\}_{i=1}^n$ and score network $\phi_{\text{final}}$.