Denoising Diffusion Probabilistic Models (DDPM) — Vault Note

0) TL;DR CARD (10 lines max)

- **Citation**: *Denoising Diffusion Probabilistic Models*, Jonathan Ho, Ajay Jain, Pieter Abbeel, NeurIPS 2020, arXiv:2006.11239v2.
- **Problem**: high-quality image generation with a likelihood-grounded model.
- **Core idea**: define a fixed **forward noising Markov chain** q and learn a **reverse denoising chain** p_θ to sample from noise back to data. (Sec. 2, p.2)
- **Key contributions**: (1) ELBO rewritten as sum of tractable Gaussian KLs (Eq. 5, Sec. 2, p.3), (2) ε-prediction parameterization (Eq. 11–12, Sec. 3.2, p.4), (3) simplified training loss L_{simple} (Eq. 14, Sec. 3.4, p.5).
- **Main results**: CIFAR-10 unconditional: IS 9.46, FID 3.17 with L_{simple} (Table 1, Sec. 4.1, p.5).
- **What's new vs prior work**: training the sampler via VI + connecting to denoising score matching; ε-parameterization + simple loss gives strong sample quality (Sec. 3.2–3.4, p.3–5).
- **Assumptions/scope**: forward schedule β_t fixed; reverse transitions Gaussian with chosen variance (Sec. 3.1–3.2, p.3–4).
- **Limitations**: slow sampling (they use $T = 1000$ steps), likelihood not SOTA vs other likelihood models (Sec. 4.3, p.6).
- **If you remember only 3 things**: (i) forward closed-form $q(x_t|x_0)$ (Eq. 4), (ii) reverse mean via noise prediction (Eq. 11), (iii) train with L_{simple} (Eq. 14).

Implementation translation

- You need: a precomputed $\beta_t, \alpha_t, \bar{\alpha}_t$ schedule; a U-Net $\varepsilon_\theta(x_t, t)$; sampling loop from $t = T \to 1$.
- Main tensor shapes: images x: $[B, C, H, W]$; timesteps t: $[B]$; noise ε: $[B, C, H, W]$.
- Numeric risk: $\sqrt{1 - \bar{\alpha}_t}$ near 0 for small t → clamp / stable dtype.

1) GLOSSARY & NOTATION (NO EXCEPTIONS)

> **Mini-explainer: "Notation table"**
> In diffusion papers, confusion usually comes from overloaded symbols (q, p, α, β). This table makes every symbol "one meaning only".
> Tiny example: if x_0 is a $32 \times 32 \times 3$ image, then x_t has the **same shape**, just noisier.

Symbol	Meaning	Shape/type	Where defined (sec/eq/page)	Notes
x_0	data sample	$\mathbb{R}^D$ (image tensor)	Sec. 2, p.2	data scaled to $[-1, 1]$ later (Sec. 3.3, p.4)
$x_{1:T}$	latents/noisy states	$\mathbb{R}^{T\times D}$	Sec. 2, Eq. 1–2, p.2	Markov chain states
$p_\theta(x_{0:T})$	reverse/generative chain joint	distribution	Sec. 2, Eq. 1, p.2	learned transitions
$p(x_T)$	prior for final noise	$\mathcal{N}(0, I)$	Sec. 2, Eq. 1, p.2	start of sampling
$p_\theta(x_{t-1} \mid x_t)$	reverse transition	Gaussian	Sec. 2, Eq. 1, p.2; Sec. 3.2, p.3–4	usually $\mathcal{N}(\mu_\theta, \Sigma_\theta)$
$q(x_{1:T} \mid x_0)$	forward/noising chain	distribution	Sec. 2, Eq. 2, p.2	fixed in main experiments
$q(x_t \mid x_{t-1})$	forward step	Gaussian	Sec. 2, Eq. 2, p.2	adds noise with variance β_t
β_t	forward variance schedule	scalar per t	Sec. 2, Eq. 2, p.2; Sec. 3.1, p.3	fixed hyperparameter in this paper
α_t	$1 - \beta_t$	scalar	Sec. 2, below Eq. 3–4, p.2	convenience
$\bar{\alpha}_t$	$\prod_{s=1}^{t} \alpha_s$	scalar	Sec. 2, near Eq. 4, p.2	cumulative signal
$q(x_t \mid x_0)$	forward marginal	Gaussian	Sec. 2, Eq. 4, p.2	key for training shortcut
L	variational bound objective	scalar	Sec. 2, Eq. 3, p.2	derived via Jensen/ELBO
L_T, L_{t-1}, L_0	decomposed ELBO terms	scalars	Sec. 2–3, Eq. 5, p.3	variance reduction
$D_{KL}(\cdot	\cdot)$	KL divergence	scalar	Eq. 5, p.3
$q(x_{t-1} \mid x_t, x_0)$	forward posterior	Gaussian	Eq. 6, p.3	tractable; used as target

Symbol	Meaning	Shape/type	Where defined (sec/eq/page)	Notes
$\tilde{\mu}_t(x_t, x_0)$	posterior mean	vector in $\mathbb{R}^D$	Eq. 7, p.3	closed form
$\tilde{\beta}_t$	posterior variance scalar	scalar	Eq. 7, p.3	closed form
σ_t^2	chosen reverse variance	scalar	Sec. 3.2, p.3	set to β_t or $\tilde{\beta}_t$
$\mu_\theta(x_t, t)$	reverse mean	$\mathbb{R}^D$	Eq. 1; Eq. 8–11, p.3–4	parameterized via ε_θ
ε	standard Gaussian noise	$\mathbb{R}^D$	Eq. 9–10, p.3	training noise
$\varepsilon_\theta(x_t, t)$	noise prediction net	$\mathbb{R}^D$	Eq. 11–14, p.4–5	main practical output
L_{simple}	simplified loss	scalar	Eq. 14, Sec. 3.4, p.5	what they actually use for best samples
Algorithm 1	training algorithm	procedure	p.4	sample t, predict noise
Algorithm 2	sampling algorithm	procedure	p.4	reverse loop $T \to 1$
IS/FID/NLL	eval metrics	scalars	Table 1, Sec. 4.1, p.5	IS↑, FID↓, NLL↓

Implementation translation

- Precompute arrays length T: $\beta_t, \alpha_t, \bar{\alpha}_t, \sqrt{\bar{\alpha}_t}, \sqrt{1-\bar{\alpha}_t}$.
- Store them as float32/float64 on device; gather by timestep indices t.
- Ensure broadcasting to image tensor: reshape gathered scalars to $[B, 1, 1, 1]$.

2) PROBLEM SETUP

2.1 Data distribution and modeling goal

- Data samples $x_0 \sim q(x_0)$ (their notation uses q for the empirical/data distribution). (Sec. 2, p.2)
- Goal: learn a generative model $p_\theta(x_0)$ that can sample realistic images and evaluate a likelihood bound. (Sec. 2, p.2)

> **Mini-explainer: latent variable model**
> A latent variable model introduces hidden variables z to make modeling easier: $p_\theta(x) = \int p_\theta(x, z), dz$
> .
> Tiny example: z could encode "pose"; x is the image. Here, the latents are the whole chain $x_{1:T}$.

2.2 Generative story (random variables + dependencies)

Reverse (generative) chain is:

$$p_\theta(x_{0:T}) := p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1} \mid x_t), \quad p(x_T) = \mathcal{N}(0, I).$$

(Eq. 1, Sec. 2, p.2)

Forward (inference/noising) chain is fixed:

$$q(x_{1:T} \mid x_0) := \prod_{t=1}^{T} q(x_t \mid x_{t-1}), \quad q(x_t \mid x_{t-1}) = \mathcal{N}\big(x_t; \sqrt{1-\beta_t}, x_{t-1}, \beta_t I\big).$$

(Eq. 2, Sec. 2, p.2)

> **Mini-explainer: Markov chain**
> "Markov" means next state depends only on current: $x_t \perp x_{0:t-2} \mid x_{t-1}$.
> Tiny example: random walk: $x_t = x_{t-1} + \text{noise}$.

2.3 What is fixed vs learned?

- Fixed: q (forward process) and usually β_t schedule (they fix it in implementation). (Sec. 3.1, p.3)
- Learned: reverse mean model $\mu_\theta(\cdot)$ (and optionally reverse variance, but they found learned diagonal variance unstable). (Sec. 3.2, p.3–4; Table 2, p.5)

Implementation translation

- Treat q as a "corruption operator" you can sample from cheaply using closed forms.
- Only neural net you need for base DDPM: $\varepsilon_\theta(x_t, t)$.
- Decide variance mode: fixed $\sigma_t^2 = \beta_t$ or $\sigma_t^2 = \tilde{\beta}_t$ (Sec. 3.2, p.3)

3) THE METHOD (PLAIN ENGLISH FIRST)

During **training**, you sample a real image x_0, pick a random timestep t, and synthesize a noisy version x_t by mixing x_0 with Gaussian noise using the closed form of the forward diffusion (Eq. 4, Sec. 2, p.2).
Then you train a neural network $\varepsilon_\theta(x_t, t)$ to predict the exact noise you injected. (Algorithm 1, p.4; Eq. 14, Sec. 3.4, p.5)
During **sampling**, you start from pure noise $x_T \sim \mathcal{N}(0, I)$ and repeatedly apply a learned reverse step to get x_{t-1} from x_t for $t = T, \ldots, 1$ (Algorithm 2, p.4).
Each reverse step is a Gaussian with mean $\mu_\theta(x_t, t)$ and variance $\sigma_t^2 I$ (Sec. 3.2, p.3–4).
The **one trick**: instead of predicting μ_θ directly, they parameterize μ_θ in terms of a noise predictor ε_θ, giving a clean MSE training objective (Eq. 11–14, Sec. 3.2–3.4, p.4–5).

> **Mini-explainer: schedule**
> The schedule β_t controls how fast signal is destroyed. Small β_t means "add a tiny bit of noise each step."
> Tiny example: if $\beta_t = 0.01$ for many steps, you slowly drift toward noise.

Implementation translation

- Training step is a single forward-noise operation + one net call + MSE.
- Sampling is T net calls; runtime scales linearly with T (they use $T = 1000$). (Sec. 4, p.5)
- Conditioning on t: sinusoidal embedding added to residual blocks (Appendix B, p.14–15).

4) MAIN EQUATIONS (THE CANONICAL SET)

Below is the **minimal sufficient** set to re-derive + implement DDPM.

Eq. 1 (Sec. 2, p.2): reverse (generative) Markov chain

$$p_\theta(x_{0:T}) := p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1} \mid x_t), \quad p(x_T) = \mathcal{N}(0, I).$$

- **Terms**: prior $p(x_T)$; transitions $p_\theta(x_{t-1} \mid x_t)$.
- **Role**: defines how sampling works (Algorithm 2).
- **Used next**: ELBO setup Eq. 3.

Eq. 2 (Sec. 2, p.2): forward (noising) Markov chain

$$q(x_{1:T} \mid x_0) := \prod_{t=1}^{T} q(x_t \mid x_{t-1}), \quad q(x_t \mid x_{t-1}) = \mathcal{N}\big(x_t; \sqrt{1-\beta_t}, x_{t-1}, \beta_t I\big).$$

- **Role**: defines corruption process; used to construct training pairs.
- **Used next**: closed form Eq. 4; ELBO Eq. 3.

Eq. 3 (Sec. 2, p.2): variational bound objective L

They write a bound on $-\log p_\theta(x_0)$:

$$\mathbb{E}\big[-\log p_\theta(x_0)\big] \le \mathbb{E}*q\left[-\log \frac{p*\theta(x_{0:T})}{q(x_{1:T}\mid x_0)}\right] =: L.$$

- **Role**: training objective derived from Jensen/ELBO (expanded in Appendix A).
- **Used next**: variance-reduced decomposition Eq. 5.

> **Mini-explainer: ELBO + Jensen**
> $\log$ is concave, so $\log \mathbb{E}[Z] \ge \mathbb{E}[\log Z]$. Rearranged, it gives a tractable lower bound on log-likelihood.
> Tiny example: for random Z, $\log(\mathrm{avg}\ Z)$ is ≥ avg($\log Z$).

Eq. 4 (Sec. 2, p.2): closed-form forward marginal

$$q(x_t \mid x_0) = \mathcal{N}\big(x_t; \sqrt{\bar\alpha_t}, x_0, (1-\bar\alpha_t)I\big), \quad \bar\alpha_t = \prod_{s=1}^{t}(1-\beta_s).$$

- **Role**: lets you sample x_t in one shot (no need to run t steps).
- **Used next**: reparameterization in Eq. 9–10.

Eq. 5 (Sec. 2, p.3): KL decomposition of ELBO

$$\mathbb{E}*q\Big[D*KL(q(x_T\mid x_0)|p(x_T)) + \sum_{t>1} D_{KL}(q(x_{t-1}\mid x_t,x_0)|p_\theta(x_{t-1}\mid x_t)) - \log p_\theta(x_0\mid x_1)\Big].$$

- **Role**: reduces Monte Carlo variance; each term is Gaussian KL or decoder term.
- **Used next**: analytic posterior Eq. 6–7 and Gaussian KL simplification Eq. 8.

Eq. 6–7 (Sec. 2, p.3): tractable posterior under forward process

$$q(x_{t-1}\mid x_t,x_0) = \mathcal{N}\big(x_{t-1}; \tilde\mu_t(x_t,x_0), \tilde\beta_t I\big),$$

$$\tilde\mu_t(x_t,x_0) = \frac{\sqrt{\bar\alpha_{t-1}}\beta_t}{1-\bar\alpha_t}x_0 + \frac{\sqrt{\alpha_t}(1-\bar\alpha_{t-1})}{1-\bar\alpha_t}x_t, \quad \tilde\beta_t = \frac{1-\bar\alpha_{t-1}}{1-\bar\alpha_t}\beta_t.$$

- **Role**: gives the “true” reverse target distribution you try to match.

Eq. 8 (Sec. 3.2, p.3): KL between Gaussians → mean MSE (if variance fixed)

$$L_{t-1} = \mathbb{E} * q \left[\frac{1}{2\sigma_t^2} \left| \tilde{\mu}_t(x_t, x_0) - \mu * \theta(x_t, t) \right|^2 \right] + C.$$

- **Role**: shows training reduces to matching means.

Eq. 11–12 (Sec. 3.2, p.4): ε-parameterization + weighted noise MSE

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

$$\mathbb{E} * x_0, \varepsilon \left[\frac{\beta_t^2}{2\sigma_t^2 \alpha_t (1 - \bar{\alpha}_t)} \left| \varepsilon - \varepsilon * \theta\big(\sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} \varepsilon, t \big) \right|^2 \right].$$

- **Role**: turns mean-matching into noise prediction.

> **Mini-explainer: reparameterization trick**
> Write a random variable as deterministic function of noise: $x = \mu + \sigma\varepsilon$. This makes gradients stable. Tiny example: sample $x \sim \mathcal{N}(\mu, 1)$ as $x = \mu + \varepsilon$.

Eq. 14 (Sec. 3.4, p.5): final training loss used for best sample quality

$$L_{\text{simple}}(\theta) := \mathbb{E} * t, x_0, \varepsilon \left[\left| \varepsilon - \varepsilon * \theta\big(\sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} \varepsilon, t \big) \right|^2 \right].$$

- **Role**: the "practical DDPM objective" used in Algorithm 1.

Algorithm 1–2 (p.4): training + sampling procedures

- **Algorithm 1**: sample t, make x_t, regress noise.
- **Algorithm 2**: reverse sampling loop using ε_θ and σ_t.

Eq. 13 (Sec. 3.3, p.4): discrete decoder for L_0

They define a discretized Gaussian decoder $p_\theta(x_0 \mid x_1)$ for integer pixels scaled to $[-1, 1]$. (Eq. 13, Sec. 3.3, p.4)

Eq. 15 (Sec. 4.3, p.7): reconstruction from predicted noise

$$\hat{x}_0 = \frac{x_t - \sqrt{1 - \bar{\alpha}_t}, \varepsilon_\theta(x_t)}{\sqrt{\bar{\alpha}_t}}.$$

- **Role**: “progressive decoding” view; also useful in later diffusion work.

Equation dependency map (fast)

- Eq. 2 $\rightarrow$ Eq. 4 $\rightarrow$ Eq. 9–10 $\rightarrow$ Eq. 11 $\rightarrow$ Eq. 12 $\rightarrow$ Eq. 14 $\rightarrow$ Algorithm 1
- Eq. 1 + Eq. 11 + chosen σ_t^2 $\rightarrow$ Algorithm 2
- Eq. 5 uses Eq. 6–7 and gives interpretation of terms L_T, L_{t-1}, L_0

Implementation translation

- Minimal implementation uses Eq. 4 and Eq. 14 only (plus Algorithm 2 for sampling).
- If you want likelihood bound terms, you need Eq. 5–8 + Eq. 13.
- Store schedule tensors; avoid recomputing $\bar{\alpha}_t$ inside training loop.

5) DERIVATION MAP (NO BIG JUMPS)

This is the “assumptions $\rightarrow$ objective $\rightarrow$ simplified loss $\rightarrow$ final training loop” chain.

Step 1 — Start from marginal likelihood (paper jump clarified)

Starting expression:

$$p_\theta(x_0) = \int p_\theta(x_{0:T}), dx_{1:T}.$$

(From model definition Eq. 1, Sec. 2, p.2)

Paper jump: They immediately introduce $q(x_{1:T} \mid x_0)$ and a variational bound.

Step 2 — Insert q and apply Jensen (the inequality step)

Write:

$$p_\theta(x_0) = \int q(x_{1:T} \mid x_0) \frac{p_\theta(x_{0:T})}{q(x_{1:T} \mid x_0)}, dx_{1:T} = \mathbb{E} * q(x * 1:T \mid x_0) \left[\frac{p_\theta(x_{0:T})}{q(x_{1:T} \mid x_0)} \right].$$

Apply $\log$ and Jensen (log is concave):

- **Identity used**: $\log \mathbb{E}[Z] \geq \mathbb{E}[\log Z]$ for $Z > 0$.

- **Result:**
  $$
  \log p_\theta(x_0)

  \log \mathbb{E}*q\left[\frac{p*\theta(x_{0:T})}{q(x_{1:T}\mid x_0)}\right]
  \ge
  \mathbb{E}*q\left[\log\frac{p*\theta(x_{0:T})}{q(x_{1:T}\mid x_0)}\right].
  $$

Multiply by -1:

$$-\log p_\theta(x_0) \le \mathbb{E} * q \left[-\log \frac{p * \theta(x_{0:T})}{q(x_{1:T} \mid x_0)}\right] =: L.$$

This is Eq. 3 (Sec. 2, p.2).

What changed and why: we replaced an intractable log-integral with a tractable expectation bound.

> **Mini-explainer: KL divergence**
> $D_{KL}(q|p) = \mathbb{E}_q[\log(q/p)]$ measures how much q differs from p (0 means identical).
> Tiny example: if q puts mass where p is tiny, KL is large.

Step 3 — Variance reduction: rewrite L as sum of KLs (Eq. 5)

Starting: L from Eq. 3.
Identity used: repeated multiplication/division to form KL terms (derivation in Appendix A, Eq. 17–22).
Result: Eq. 5 decomposition (Sec. 2, p.3).

Concretely, Appendix A shows:

$$L = \mathbb{E} * q \left[D * KL(q(x_T \mid x_0)|p(x_T)) + \sum_{t>1} D_{KL}(q(x_{t-1} \mid x_t, x_0)|p_\theta(x_{t-1} \mid x_t)) - \log p_\theta(x_0 \mid x_1)\right]$$

(Eq. 22 in Appendix A, p.14; matches Eq. 5 in main text p.3).

What changed and why: we turned one big expectation into per-step terms with lower-variance estimation (Rao–Blackwellization claim in text). (Sec. 2, p.3)

Step 4 — Make each KL tractable: use Gaussian posterior (Eq. 6–7)

Starting: KLs in Eq. 5 include $q(x_{t-1} \mid x_t, x_0)$.

Identity used: Gaussian conditioning of a linear-Gaussian Markov chain.

Result: Eq. 6–7 (Sec. 2, p.3).

Step 5 — Gaussian KL gives mean MSE (Eq. 8)

Assume reverse variance fixed: $\Sigma_\theta = \sigma_t^2 I$ (Sec. 3.2, p.3).
Then KL between Gaussians reduces to:

$$L_{t-1} = \mathbb{E} * q \left[\frac{1}{2\sigma_t^2} |\tilde{\mu}_t - \mu * \theta|^2 \right] + C$$

(Eq. 8, Sec. 3.2, p.3).

Step 6 — Reparameterize x_t using Eq. 4 (Eq. 9–10)

Starting: Eq. 8 expectation over $q(x_t \mid x_0)$.

Identity used: reparameterize Gaussian:

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

(Used to go from Eq. 8 $\to$ Eq. 9, Sec. 3.2, p.3).

Result: Eq. 9 then simplified to Eq. 10 (Sec. 3.2, p.3).

Step 7 — Key observation: the target mean contains ε (Eq. 10 → Eq. 11)

Eq. 10 shows the target structure:

$$\frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}} \varepsilon \right).$$

(Sec. 3.2, Eq. 10, p.3–4).

So they choose:

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

(Eq. 11, Sec. 3.2, p.4).

What changed and why: instead of predicting μ_θ directly, predict ε_θ so the regression target is “always noise”.

Step 8 — Substitute Eq. 11 into Eq. 10: mean MSE becomes noise MSE (Eq. 12)

After substitution, the x_t terms cancel and you get a weighted noise prediction objective:

$$\mathbb{E} * x_0, \varepsilon \left[\frac{\beta_t^2}{2\sigma_t^2 \alpha_t (1 - \bar{\alpha}_t)} |\varepsilon - \varepsilon * \theta(\sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} \varepsilon, t)|^2 \right].$$

(Eq. 12, Sec. 3.2, p.4).

Step 9 — Drop the weight: define L_{simple} (Eq. 14)

They report best sample quality using:

$$L_{\text{simple}}(\theta) = \mathbb{E} * t, x_0, \varepsilon \left[|\varepsilon - \varepsilon * \theta(\sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} \varepsilon, t)|^2 \right].$$

(Eq. 14, Sec. 3.4, p.5).

What becomes constant / dropped and why:

- They fix β_t, so L_T becomes constant during training (Sec. 3.1, p.3).
- L_{simple} discards the weight from Eq. 12; they argue it improves sample quality and shifts focus to larger t (Sec. 3.4, p.5).

Implementation translation

- Re-derivation you should be able to redo: Eq. 8 → Eq. 10 → Eq. 11 → Eq. 12 → Eq. 14.
- Only inequality is Jensen in Step 2; everything else is algebra + Gaussian identities.
- Practical training ignores L_T and uses L_{simple} + Algorithm 1.

6) OBJECTIVE / LOSS (FINAL FORM + INTERPRETATION)

6.1 Full bound they start from (Eq. 3)

$$L = \mathbb{E} * q \left[-\log \frac{p * \theta(x_{0:T})}{q(x_{1:T} \mid x_0)} \right]$$

(Eq. 3, Sec. 2, p.2).

Interpretation: minimizing L approximately maximizes likelihood.

6.2 Decomposed bound (Eq. 5) and what each term does

$$\mathbb{E} * q \Big[D * KL(q(x_T \mid x_0) | p(x_T)) + \sum_{t>1} D_{KL}(q(x_{t-1} \mid x_t, x_0) | p_\theta(x_{t-1} \mid x_t)) - \log p_\theta(x_0 \mid x_1) \Big].$$

(Eq. 5, Sec. 2, p.3).

- L_T **term**: matches terminal noisy marginal to prior; with fixed q, constant in training (Sec. 3.1, p.3).
- L_{t-1} **terms**: train reverse transitions to match true posteriors (Sec. 3.2, p.3).
- L_0 **term**: decoder likelihood for discrete pixels (Sec. 3.3, Eq. 13, p.4).

> **Mini-explainer: "posterior"**
> Posterior means "distribution of hidden variables given what you observed." Here: $q(x_{t-1} \mid x_t, x_0)$.
> Tiny example: if you observe noisy x_t, posterior tells you plausible cleaner x_{t-1}.

6.3 What they actually train for best samples: L_{simple} (Eq. 14)

$$L_{\text{simple}}(\theta) = \mathbb{E} * t, x_0, \varepsilon \left[|\varepsilon - \varepsilon * \theta(\sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} \varepsilon, t)|^2 \right].$$

(Eq. 14, Sec. 3.4, p.5).

Weighted vs unweighted tradeoff

- Eq. 12 has a weight $w_t = \frac{\beta_t^2}{2\sigma_t^2 \alpha_t (1 - \bar{\alpha}_t)}$ (Sec. 3.2, p.4).
- Eq. 14 drops w_t; they argue this reweights which noise levels matter and improves sample quality (Sec. 3.4, p.5).

Ambiguity note (what "$t = 1$ corresponds to L_0" means)

- They state: "$t = 1$ corresponds to L_0 with the integral approximated…" (Sec. 3.4, p.5).

- **Possibility A**: for $t = 1$, the ε-MSE approximates discretized decoder training (evidence: their text).
- **Possibility B**: it's a heuristic replacement, not exactly the same as Eq. 13 (evidence: they mention ignoring σ_1^2 and edge effects).

Implementation translation

- If you just want SOTA-like sampling: implement Eq. 14 + Algorithm 2.
- If you want bits/dim: implement Eq. 5 + Eq. 13; that's extra work.
- Watch small t: $1 - \bar{\alpha}_t$ tiny → noise scale tiny → gradients can be small.

7) ALGORITHMS (TRAINING + SAMPLING)

7.1 Training loop pseudocode (Algorithm 1, p.4)

Inputs: dataset samples x_0, schedule arrays, network ε_θ.

```
1. Sample minibatch $x_0 \sim q(x_0)$
2. Sample timesteps $t \sim \mathrm{Uniform}(1, \ldots, T)$
3. Sample noise $\varepsilon \sim \mathcal{N}(0, I)$
4. Form $x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} \varepsilon$
5. Minimize $|\varepsilon - \varepsilon_\theta(x_t, t)|^2$ (Eq. 14)
```

(Algorithm 1; Eq. 14; p.4–5).

7.2 Sampling pseudocode (Algorithm 2, p.4)

Inputs: T, schedule arrays, variance choice σ_t^2, network ε_θ.

```
1. Sample $x_T \sim \mathcal{N}(0, I)$
2. For $t = T, \ldots, 1$:
    • Sample $z \sim \mathcal{N}(0, I)$ if $t > 1$ else $z = 0$
    • Compute mean using Eq. 11
    • Sample $x_{t-1} = \mu_\theta(x_t, t) + \sigma_t z$
3. Return $x_0$
```

(Algorithm 2, p.4).

7.3 Minimal PyTorch-like skeleton (structure only)

```
# precompute beta, alpha, abar as torch tensors shape [T]
# store sqrt_abar, sqrt_one_minus_abar

def q_sample(x0, t, noise):
    # gather scalars -> [B,1,1,1]
    return sqrt_abar[t] * x0 + sqrt_1m_abar[t] * noise

def train_step(model_eps, x0_batch):
    B = x0_batch.shape[0]
    t = torch.randint(1, T+1, (B,), device=x0_batch.device)  # 1..T
    eps = torch.randn_like(x0_batch)
    xt = q_sample(x0_batch, t, eps)
    eps_hat = model_eps(xt, t)
    loss = (eps - eps_hat).pow(2).mean()
    return loss

@torch.no_grad()
def sample(model_eps):
    x = torch.randn((B,C,H,W), device=device)  # x_T
    for t in range(T, 0, -1):
        z = torch.randn_like(x) if t > 1 else torch.zeros_like(x)
        eps_hat = model_eps(x, torch.full((B,), t, device=device))
        mu = (1 / torch.sqrt(alpha[t])) * (x - (beta[t] / torch.sqrt(1-abar[t])) * eps_hat)
        x = mu + sigma[t] * z
    return x
```

Implementation translation

- Required tensors: `beta[t], alpha[t], abar[t], sqrt_abar[t], sqrt_1m_abar[t], sigma[t]` .
- Shapes: gather scalars using indexing then reshape to broadcast: `[B,1,1,1]` .
- Numerical issues: `1-abar[t]` can underflow for large t in float16; use float32.

8) DESIGN CHOICES & ABLATIONS

Choice	Options tried	What changed	Effect on results	My takeaway
Reverse variance σ_t^2	$\sigma_t^2 = \beta_t$ vs $\sigma_t^2 = \tilde{\beta}_t$	sampling noise magnitude	similar results (Sec. 3.2, p.3)	either is fine; many later works use $\tilde{\beta}_t$

Choice	Options tried	What changed	Effect on results	My takeaway
Predict target	predict $\tilde{\mu}_t$ vs predict ε	network output meaning	ε + L_{simple} best FID (Table 2, p.5)	ε-prediction is the practical standard
Learn variance	learned diagonal Σ_θ vs fixed isotropic	extra head + KL term	unstable / poorer samples (Table 2, p.5)	keep variance fixed in baseline DDPM
Objective	full L vs L_{simple}	weighting of timesteps	L_{simple} gives best sample quality (Sec. 3.4, p.5; Table 1)	optimize for samples, not necessarily bits/dim
Schedule	constant/linear/quadratic; chose linear	β_t shape	chose linear $\beta_1 = 10^{-4}$ to $\beta_T = 0.02$ (Sec. 4, p.5; Appendix B, p.15)	schedule strongly affects training stability

Implementation translation

- "Predict ε" means model output shape equals image shape.
- If you try learned variance, you must implement Gaussian KL exactly; expect instability like paper.
- Schedule must keep $L_T \approx 0$ (they constrain this in sweeps).

9) IMPLEMENTATION & REPRODUCTION NOTES

9.1 Datasets + preprocessing

- Pixel values are integers in $0, \ldots, 255$ scaled linearly to $[-1, 1]$ (Sec. 3.3, p.4).
- CIFAR-10 and CelebA-HQ loaded via TFDS; LSUN prepared using StyleGAN code (Appendix B, p.15).

9.2 Model architecture (Appendix B)

- Backbone: U-Net like PixelCNN++ / Wide ResNet; group norm instead of weight norm (Appendix B, p.14).
- Resolutions: 4 levels for 32×32; 6 levels for 256×256 (Appendix B, p.14).
- 2 residual blocks per resolution; self-attention at 16×16 (Appendix B, p.14).

- Time embedding: Transformer sinusoidal position embedding added into each residual block (Appendix B, p.14).
- Params: CIFAR model 35.7M; LSUN/CelebA-HQ 114M (Appendix B, p.14).

9.3 Optimization + training details

- $T = 1000$ (Sec. 4, p.5).
- Linear schedule $\beta_1 = 10^{-4}$ to $\beta_T = 0.02$ (Sec. 4, p.5; Appendix B p.15).
- Optimizer: Adam, “standard values”; LR 2×10^{-4} for CIFAR, 2×10^{-5} for 256^2 (Appendix B, p.15).
- Batch size: 128 for CIFAR; 64 for larger images (Appendix B, p.15).
- EMA: decay 0.9999 (Appendix B, p.15).
- Dropout: CIFAR 0.1; others 0 (Appendix B, p.15).
- Augment: random horizontal flips for CIFAR and others except LSUN Bedroom (Appendix B, p.15).

9.4 Compute / runtime

- TPU v3-8 used (Appendix B, p.14).
- CIFAR: 21 steps/s at batch 128; 800k steps ~10.6h; sampling 256 images ~17s (Appendix B, p.14).
- 256x256: 2.2 steps/s at batch 64; sampling 128 images ~300s (Appendix B, p.14).

9.5 Gotchas & stability notes

- Keep β_t small so forward/reverse have similar functional form (Sec. 4, p.5).
- For t indexing: paper uses $t \in 1, \ldots, T$; be consistent in code.
- Use float32 for schedule computations to avoid underflow of $\bar{\alpha}_t$.
- Learning Σ_θ can destabilize training (Table 2, p.5).
- Data scaling to $[-1, 1]$ matters for matching the Gaussian prior scale (Sec. 3.3, p.4).

9.6 Hyperparameters that matter most (ranked)

1. β_t schedule shape + endpoints (Sec. 4, p.5; Appendix B, p.15).
2. Objective weighting (full L vs L_{simple}) (Sec. 3.4, p.5).
3. Variance choice σ_t^2 (Sec. 3.2, p.3).
4. Architecture capacity + attention placement (Appendix B, p.14).

Implementation translation

- Repro = schedule + U-Net + time embedding + EMA + dropout (CIFAR) + L_{simple}.
- Use fixed isotropic variance first; only then try fancy changes.
- Evaluate with FID on 50k samples as they do (Appendix B, p.15).

10) RESULTS & EVALUATION

10.1 Metrics

- Inception Score (IS): higher is better.
- FID: lower is better.
- NLL in bits/dim: lower is better (reported as a bound / codelength). (Table 1, p.5)

10.2 Main results (high-signal numbers)

- CIFAR-10 unconditional, L_{simple}: IS 9.46 ± 0.11, FID 3.17, NLL ≤ 3.75 bits/dim (Table 1, p.5).
- With full L (fixed isotropic variance): FID 13.51 (Table 1, p.5).
- LSUN FIDs (256x256): Bedroom 4.90 (large model), Church 7.89, Cat 19.75 (Table 3, p.13).

10.3 Baselines + fairness

- They compare against GANs, autoregressive models, and score-based models (Table 1).
- Note: many baselines differ in compute and conditioning (Table 1 includes conditional models too), so treat comparisons as “ballpark” unless compute matched.

10.4 Likelihood reporting

- NLL is reported in bits/dim; decoder uses discretized Gaussian formulation (Eq. 13, Sec. 3.3, p.4).
- They mention their likelihoods are not competitive with best likelihood models (Sec. 4.3, p.6).

Implementation translation

- If you want to match their reported sample metrics: generate 50k samples, compute IS/FID in standard pipelines (Appendix B).
- Sampling is slow: at 256^2, 128 images ~300s on TPU v3-8 (Appendix B).

11) INTUITION & CONNECTIONS

11.1 Mechanistic intuition (why it works)

- The forward process destroys information gradually in a controlled way (Eq. 2–4).
- The reverse step only needs to solve a **small denoising problem** at each t instead of modeling data directly.
- Predicting ε makes the target “stationary-ish”: the network always predicts Gaussian noise, not a moving mean target (Eq. 11–14).

> **Mini-explainer: "score"**
> The score of a distribution is $\nabla_x \log p(x)$ (gradient of log-density). It points toward higher-probability regions.
> Tiny example: if $p(x) = \mathcal{N}(0,1)$, then $\nabla_x \log p(x) = -x$.

11.2 What is "denoising score matching" here?

They say Eq. 12 "resembles denoising score matching over multiple noise scales" (Sec. 3.2, p.4).

Bridge idea (high level):

- For Gaussian corruption $x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1-\bar{\alpha}_t}\varepsilon$, the noise ε is proportional to the score of the corrupted density in certain formulations; so learning ε_θ is closely related to learning a score function at noise level t. (That's the "connection" they highlight.)

> **Mini-explainer: score matching**
> Score matching fits a model $s_\theta(x)$ to approximate $\nabla_x \log p(x)$ without needing $p(x)$ normalized.
> Tiny example: learn $s_\theta(x)$ so that $s_\theta(x) \approx -x$ for standard normal data.

11.3 Connections they explicitly mention

- Connection to denoising score matching + annealed Langevin dynamics (Sec. 3.2, p.4; Related Work p.8).
- Reverse sampling resembles Langevin dynamics (Algorithm 2 discussion, p.4).
- Autoregressive decoding interpretation via alternate ELBO form (Eq. 16, Sec. 4.3, p.7–8; Appendix A Eq. 23–26, p.14).

Implementation translation

- You do not need score matching theory to implement DDPM, but it explains why ε-prediction is natural.
- If you later read score-based SDE papers, Eq. 12 is your bridge.
- Algorithm 2 is "Langevin-like" because it adds Gaussian noise each step.

12) LIMITATIONS, ASSUMPTIONS, AND OPEN QUESTIONS

12.1 Explicit limitations / scope

- Sampling cost: $T = 1000$ reverse steps (Sec. 4, p.5).
- Likelihood (bits/dim) not competitive with best likelihood models (Sec. 4.3, p.6).
- Forward process β_t fixed in their implementation (Sec. 3.1, p.3).

12.2 Implicit limitations (inference)

- If β_t too large, Gaussian reverse may not approximate well (they emphasize small β_t for reversibility). (Sec. 4, p.5; Appendix C point 3, p.15).
- Discrete decoder L_0 is relatively simple; stronger decoders could improve likelihood (Sec. 3.3, p.4).

12.3 Open questions / what I'd test next

1. Replace L_{simple} with principled timestep weighting: does it keep FID while improving bits/dim?
2. How sensitive is FID to $\sigma_t^2 = \beta_t$ vs $\tilde{\beta}_t$ across datasets?
3. Can learned variance be stabilized with constraints/clipping?
4. Can we reduce steps T without losing quality (fast sampling)?
5. Does predicting x_0 (they said worse early) improve with better weighting/architecture? (Sec. 3.2, p.4).

Implementation translation

- Main "break" mode: schedule too aggressive → reverse chain struggles.
- Main "cost" mode: too many steps → slow.
- Best first experiment: try different schedules while keeping everything else fixed.

13) "STEAL THIS" SECTION (PORTABLE IDEAS)

1. **ELBO decomposition into per-step KLs**
    - Where: Eq. 5 (Sec. 2, p.3).
    - Why: lower-variance training; interpretable per-timestep terms.
    - Apply elsewhere: any latent Markov model with tractable conditionals.
2. **ε-parameterization of reverse mean**
    - Where: Eq. 11–12 (Sec. 3.2, p.4).
    - Why: turns learning into stable noise regression.
    - Apply elsewhere: any "denoise to clean" model under Gaussian corruption.
3. **Train with random timestep sampling**
    - Where: Algorithm 1 + Eq. 14 (p.4–5).
    - Why: covers all noise scales; easy SGD.
    - Apply elsewhere: multi-noise-level denoisers / score models.
4. **Fixed variance for stability**
    - Where: Sec. 3.2 + Table 2 (p.3–5).
    - Why: learned variance was unstable in their experiments.
    - Apply elsewhere: start with fixed variance; learn later.

Implementation translation

- These 4 tricks are enough to build a strong baseline DDPM.
- Steal order: (schedule + Eq. 4) → Eq. 14 training → Eq. 11 sampling.
- Keep a “schedule module” separate so you can swap linear/cosine later.

14) SELF-TEST (FOR LEARNING)

10 short questions

1. What does $q(x_t \mid x_{t-1})$ do?
2. What are α_t and $\bar{\alpha}_t$?
3. What is the difference between q and p_θ?
4. What does the ELBO bound (Eq. 3) guarantee?
5. Why is Eq. 4 important for training speed?
6. What distribution is $p(x_T)$?
7. What is $\varepsilon_\theta(x_t, t)$ predicting?
8. Why does fixing β_t make L_T constant?
9. What is the role of σ_t^2 in sampling?
10. What does FID measure directionally (higher/lower)?

5 medium questions

1. Derive Eq. 4 from repeated application of Eq. 2 (at a high level).
2. Show how Jensen produces Eq. 3 from the marginal likelihood.
3. Explain why Gaussian KL reduces to mean MSE in Eq. 8 when variance fixed.
4. Starting from Eq. 10, motivate Eq. 11 (why ε shows up).
5. Explain the difference between Eq. 12 and Eq. 14 and why Eq. 14 can help samples.

2 hard questions

1. If you replace uniform t sampling with a custom distribution, how would you adjust the loss to keep the same objective?
2. Suppose you set β_t much larger so x_T is “very noisy” quickly. Predict failure modes for sampling and training.

Answers (collapsible)

▶ Short answers

▶ Medium + hard answers (sketches)

Implementation translation

- Self-test items 4–8 map directly to code: you should be able to point at where each appears in the training loop.
- If you can re-derive Eq. 11 from Eq. 10, you “own” DDPM.

15) FINAL CHECKLIST (MUST INCLUDE)

- [x] I listed the canonical equations and explained each
- [x] I made a derivation map with no big jumps (Jensen + substitutions shown)
- [x] I gave sampling + training pseudocode and a minimal PyTorch-like skeleton
- [x] I extracted experimental/repro details (Appendix B + Sec. 4)
- [x] I summarized results + ablations + limitations (Tables 1–3, Sec. 4)
- [x] I wrote a complete notation glossary with shapes
- [x] I included gotchas + ranked hyperparameters + what to test next

If you want, I can also generate a **second vault note** that is only the “implementation core” (one screen long): *schedule precompute* + *training step* + *sampling step* + *common pitfalls*.