Denoising Diffusion Probabilistic Models — Vault Note

0) TL;DR CARD (10 lines max)

- Citation: Ho, Jain, Abbeel. *Denoising Diffusion Probabilistic Models*. NeurIPS 2020. arXiv:2006.11239v2.
- Problem (1–2 lines): Learn a generative model for images that can both **sample** high-quality images and **evaluate** (a bound on) log-likelihood.
- Core idea (2–4 lines): Define a **fixed forward** noising Markov chain $q(x_{1:T} \mid x_0)$ and learn a **reverse** denoising Markov chain $p_\theta(x_{0:T})$ that inverts it. Train via a variational bound that becomes (almost) a **noise-prediction MSE** when reparameterized.
- Key contributions (≤3 bullets): (i) Variational bound rewritten into low-variance KL terms (Eq. 5, Sec. 2, p.3). (ii) Parameterize reverse mean via **ϵ-prediction** (Eq. 11, Sec. 3.2, p.4) making the loss resemble multi-noise denoising score matching (Eq. 12, Sec. 3.2, p.4). (iii) Use simplified objective L_{simple} (Eq. 14, Sec. 3.4, p.5) for best sample quality.
- Main results: CIFAR-10: IS 9.46 ± 0.11, FID 3.17, NLL ≤ 3.75 bits/dim (Table 1, p.5).
- What's actually new vs prior work: the **ϵ-prediction parameterization + objective simplification** that makes training stable + high-quality sampling with a clean derivation from the diffusion ELBO.
- Assumptions / scope: Gaussian diffusion, fixed forward variances β_t, reverse transitions Gaussian with fixed isotropic variance $\sigma_t^2 I$.
- When it fails / limitations: slow sampling (typically $T = 1000$ reverse steps), likelihood not SOTA vs top autoregressive models.
- "If you remember only 3 things": (1) Forward: $x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$ (Eq. 4, Sec. 2, p.2). (2) Learn $\epsilon_\theta(x_t, t)$, not x_0 directly (Eq. 11–14, Sec. 3.2–3.4, p.4–5). (3) Sample by iterating the closed-form reverse update (Alg. 2, p.4).

Implementation translation

- Tensors: images $x_0 \in [-1, 1]^{B \times C \times H \times W}$, noises $\epsilon, z \sim \mathcal{N}(0, I)$ same shape.
- Precompute arrays (length T): $\beta_t, \alpha_t, \bar{\alpha}_t, \sqrt{\bar{\alpha}_t}, \sqrt{1-\bar{\alpha}_t}, 1/\sqrt{\alpha_t}, \beta_t/\sqrt{1-\bar{\alpha}_t}$.
- Numerics: compute $\bar{\alpha}_t = \prod_{s \leq t} \alpha_s$ in float64 (or log space) to avoid underflow for large T.

1) GLOSSARY & NOTATION (NO EXCEPTIONS)

> Mini-explainer: **Posterior** $p(z \mid x)$ is "what latent z likely was, given observed x." Example: if $x = z + \text{noise}$, posterior concentrates near $z \approx x$.
> Mini-explainer: **KL divergence** $D_{KL}(q|p)$ measures how much q differs from p (0 if identical). Example: KL between $\mathcal{N}(0,1)$ and $\mathcal{N}(1,1)$ equals $\frac{1}{2}$.

Symbol	Meaning	Shape/type	Where defined (sec/eq/page)	Notes
x_0	data sample (image)	$\mathbb{R}^D$ or $\mathbb{R}^{C\times H\times W}$	Sec. 2, Eq. 1–3, p.2	Data scaled to $[-1,1]$ (Sec. 3.3, p.4).
$q(x_0)$	data distribution	empirical dataset	Alg. 1, p.4	"Sample a training image."
D	dimensionality (pixels $\times$ channels)	int	Eq. 13, Sec. 3.3, p.4–5	Used in discrete decoder product.
t	diffusion timestep	int in $1,\dots,T$	Alg. 1, p.4	Sampled uniformly in L_{simple}.
T	number of diffusion steps	hyperparam int	Sec. 4, p.5; App. B, p.15	They fix $T = 1000$.
$x_{1:T}$	latent/noised variables	same shape as x_0	Sec. 2, Eq. 1–3, p.2–3	Markov chain forward/backward.
$q(x_{1:T} \mid x_0)$	forward noising process	Markov chain	Sec. 2, Eq. 2–3, p.2	Fixed (no learned params).
$q(x_t \mid x_{t-1})$	forward step	Gaussian	Sec. 2, Eq. 2, p.2	Adds noise with variance β_t.
β_t	forward variance schedule	float (per t)	Sec. 4, p.5; App. B, p.15	Linear $\beta_1 = 10^{-4}$ to $\beta_T = 0.02$.
α_t	$1 - \beta_t$	float	Eq. 4, Sec. 2, p.2	Convenience.

Symbol	Meaning	Shape/type	Where defined (sec/eq/page)	Notes
$\bar{\alpha}_t$	$\prod_{s=1}^{t} \alpha_s$	float	Eq. 4, Sec. 2, p.2	Cumulative signal retention.
$q(x_t \mid x_0)$	closed-form marginal	Gaussian	Eq. 4, Sec. 2, p.2	$x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$.
ϵ	standard Gaussian noise	same shape as x_0	Eq. 9–12, Sec. 3.2, p.4	Reparameterization trick.
$p_\theta(x_{0:T})$	learned reverse generative model	Markov chain	Sec. 2, Eq. 1, p.2	Factorizes as prior $p(x_T)$ and reverse transitions.
$p(x_T)$	prior on final latent	$\mathcal{N}(0, I)$	Sec. 2, Eq. 1, p.2	Matches $q(x_T \mid x_0)$ approx.
$p_\theta(x_{t-1} \mid x_t)$	reverse step	Gaussian	Sec. 2 / 3.2, p.2–4	$\mathcal{N}(\mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$.
$\mu_\theta(x_t, t)$	reverse mean network output (or derived from ϵ_θ)	tensor like x_t	Eq. 11, Sec. 3.2, p.4	Key design choice.
$\Sigma_\theta(x_t, t)$	reverse covariance	matrix or diag	Sec. 3.2, p.3–4	They fix isotropic: $\Sigma_\theta = \sigma_t^2 I$.
σ_t^2	fixed reverse variance	float	Sec. 3.2, p.3	Try $\sigma_t^2 = \beta_t$ or $\tilde{\beta}_t$.
$q(x_{t-1} \mid x_t, x_0)$	forward posterior (tractable)	Gaussian	Eq. 6, Sec. 2, p.3	Used inside KLs in Eq. 5.
$\tilde{\mu}_t(x_t, x_0)$	posterior mean of $q(x_{t-1} \mid x_t, x_0)$	tensor like x_t	Eq. 7, Sec. 2, p.3	Closed form.
$\tilde{\beta}_t$	posterior variance	float	Eq. 7, Sec. 2, p.3	$\tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$.

Symbol	Meaning	Shape/type	Where defined (sec/eq/page)	Notes
L	variational objective (ELBO-style)	scalar	Eq. 3, Sec. 2, p.2	Lower-bounds $-\log p_\theta(x_0)$.
L_T	terminal KL term	scalar	Eq. 5, Sec. 2, p.3	$D_{KL}(q(x_T\mid x_0)
L_{t-1}	per-step KL term	scalar	Eq. 5, Sec. 2, p.3	$D_{KL}(q(x_{t-1}\mid x_t,x_0)
L_0	decoder / reconstruction term	scalar	Eq. 5, Sec. 2, p.3	$-\log p_\theta(x_0 \mid x_1)$.
L_{simple}	simplified training loss	scalar	Eq. 14, Sec. 3.4, p.5	Unweighted MSE on ϵ.
$\epsilon_\theta(x_t, t)$	network predicts noise	tensor like x_t	Eq. 11–14, Sec. 3.2–3.4, p.4–5	The “one trick.”
z	sampling noise at each reverse step	tensor like x_t	Alg. 2, p.4	$z = 0$ when $t = 1$.
$\delta^+(x), \delta^-(x)$	bin edges for discrete decoder	scalar funcs	Eq. 13, Sec. 3.3, p.4–5	Handles $[-1, 1]$ edge cases.
$H(x_0)$	entropy of data dist	scalar const	Eq. 26, App. A, p.14	Appears in autoregressive connection.
$D_{KL}(\cdot	\cdot)$	KL divergence	scalar	Eq. 5, p.3; Eq. 22, App. A, p.14

Implementation translation

- Shapes: treat everything as $[B, C, H, W]$; $D = C \cdot H \cdot W$ if flattened.

- Time t: store as int64 tensor $[B]$; embed to $[B, d_{\text{emb}}]$ (sinusoidal) and broadcast into residual blocks (App. B, p.15).
- Distributions: you never explicitly construct Gaussians for L_{simple}—just sample ϵ and compute MSE.

2) PROBLEM SETUP

Goal

Model the unknown data distribution $q(x_0)$ over images by learning $p_\theta(x_0)$ such that:

1. you can **sample** $x_0 \sim p_\theta$, and
2. you can **evaluate** a tractable bound related to $-\log p_\theta(x_0)$ via a variational objective (Sec. 2, Eq. 3–5, p.2–3).

Generative story (random variables + dependencies)

1. Sample a latent "pure noise" variable $x_T \sim p(x_T) = \mathcal{N}(0, I)$ (Sec. 2, Eq. 1, p.2).
2. For $t = T, T-1, \ldots, 1$, sample

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

   (Sec. 3.2, p.3–4).
3. Output x_0 (optionally through a discrete decoder $p_\theta(x_0 \mid x_1)$, Eq. 13, Sec. 3.3, p.4–5).

What is assumed known / fixed vs learned?

- **Fixed/known:** forward process q and schedule $\beta_t{}_{t=1}^{T}$ (Sec. 3.1, p.3; Sec. 4, p.5).
- **Learned:** reverse mean function (implemented by a neural net, usually as ϵ_θ) and optionally variance (but learning variance was unstable; Sec. 4.2, p.6; Table 2, p.5).
- **Reverse variance choice:** fixed isotropic $\Sigma_\theta = \sigma_t^2 I$ with $\sigma_t^2 \in \beta_t, \tilde{\beta}_t$ (Sec. 3.2, p.3).

> Mini-explainer: A **schedule** (here β_t) is just a planned sequence controlling how much noise is added each step.
> Tiny example: if $T = 3$ and $\beta = [0.1, 0.2, 0.3]$, later steps inject more noise than early steps.

Implementation translation

- Forward “noising” for training uses the closed form (Eq. 4, Sec. 2, p.2) so you don’t loop over t: sample one t, one ϵ, form x_t directly.
- Fixed vs learned: only network parameters in $\epsilon_\theta(\cdot, \cdot)$ get gradients under L_{simple} (Eq. 14, Sec. 3.4, p.5).

3) THE METHOD (PLAIN ENGLISH FIRST)

During **training**, you take a real image x_0, pick a random timestep t, and create a noisy version x_t by mixing x_0 with Gaussian noise using the known scalar coefficients $\sqrt{\bar{\alpha}_t}$ and $\sqrt{1 - \bar{\alpha}_t}$ (Eq. 4, Sec. 2, p.2).
You then train a U-Net to predict the exact noise ϵ that was used to corrupt x_0 into x_t (Eq. 14, Sec. 3.4, p.5; Alg. 1, p.4).
The reason this works is that, for Gaussian chains, the reverse transition that best inverts the forward process can be written in closed form using the *posterior* $q(x_{t-1} \mid x_t, x_0)$, and matching that posterior reduces to a squared error between means when variances are fixed (Eq. 6–8, Sec. 2–3.2, p.3–4).
By rewriting the reverse mean μ_θ in terms of a predicted noise ϵ_θ, the complicated KL-based term becomes a weighted MSE on ϵ (Eq. 11–12, Sec. 3.2, p.4).
They finally drop the weight (a “weighted ELBO” variant) to get the simplest and best-performing loss L_{simple} (Eq. 14, Sec. 3.4, p.5).

During **sampling**, you start from pure Gaussian noise $x_T \sim \mathcal{N}(0, I)$ and repeatedly apply the learned denoising step for $t = T \to 1$ (Alg. 2, p.4).
Each step uses $\epsilon_\theta(x_t, t)$ to compute a mean (via Eq. 11) and adds fresh Gaussian noise with variance σ_t^2 (Sec. 3.2, p.3; Alg. 2, p.4).
The “one trick” is: **predict the noise** ϵ (not x_0 or $\tilde{\mu}_t$) so training is a straightforward denoising regression problem that lines up with the reverse-process KL objective (Eq. 11–12, p.4).

> Mini-explainer: **ELBO** is a lower bound on log-likelihood used to train latent-variable models. You optimize an easier bound instead of the exact $\log p_\theta(x)$.
> Tiny example: in a VAE, ELBO = reconstruction term − KL to prior; here ELBO decomposes into many per-timestep KLs (Eq. 5, Sec. 2, p.3).

Implementation translation

- Network I/O: input (x_t, t), output ϵ_θ same shape as x_t (Alg. 1 step 5, p.4).
- Sampling loop: compute x_{t-1} with precomputed scalars; add noise except at $t = 1$ (Alg. 2, p.4).

- Numerical issues: clamp or carefully handle divisions by $\sqrt{1-\bar{\alpha}_t}$ when t small; also float precision for $\bar{\alpha}_t$.

4) MAIN EQUATIONS (THE CANONICAL SET)

Below is a "minimal sufficient" set to rebuild training + sampling.

Eq. (1) — Reverse process factorization (Sec. 2, p.2)

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

- Each term: $p(x_T)$ prior; reverse transitions are Gaussian with learned mean.
- Why it matters: defines the generative model and sampling procedure (Alg. 2, p.4).
- Used next: inside ELBO (Eq. 3, Sec. 2, p.2) and KL decomposition (Eq. 5, Sec. 2, p.3).

Eq. (2) — Forward diffusion/noising step (Sec. 2, p.2)

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

- Each term: β_t controls noise strength per step.
- Why it matters: provides a *known* corruption process to invert.
- Used next: yields closed-form marginal (Eq. 4).

Eq. (3) — Variational training objective (ELBO form) (Sec. 2, p.2–3)

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

- Each term: expectation under $q(x_0)q(x_{1:T} \mid x_0)$.
- Why it matters: optimizing L trains p_θ (lower-bounds NLL).
- Used next: rewritten to low-variance KL sum (Eq. 5; App. A Eq. 17–22).

Eq. (4) — Closed-form $q(x_t \mid x_0)$ (Sec. 2, p.2)

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

- Why it matters: lets you sample x_t in one shot for random t.
- Used next: (i) posterior $q(x_{t-1} \mid x_t, x_0)$ (Eq. 6–7), (ii) reparameterization $x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$ (Eq. 9, Sec. 3.2, p.4).

Eq. (5) — KL decomposition of L (Sec. 2, p.3)

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

- Why it matters: turns ELBO into a sum of **Gaussian KLs** + decoder term, lowering variance.
- Used next: define L_T, L_{t-1}, L_0; analyze L_{t-1} (Eq. 8–12).

Eq. (6)–(7) — Forward posterior is Gaussian (Sec. 2, p.3)

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

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

- Why it matters: gives the “target” reverse step distribution.
- Used next: KL between Gaussians simplifies (Eq. 8).

Eq. (8) — KL term reduces to squared error in means (Sec. 3.2, p.3–4)

With $p_\theta(x_{t-1} \mid x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \sigma_t^2 I)$:

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

- Why it matters: training becomes regression if variance fixed.
- Used next: reparameterize using ϵ to get Eq. 12.

Eq. (11) — Mean parameterization via ϵ_θ (Sec. 3.2, p.4)

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

- Each term: subtract estimated noise then rescale.
- Why it matters: makes sampling update simple (Alg. 2 line 4).
- Used next: plug into Eq. 8 → Eq. 12.

Eq. (12) — Weighted MSE on noise (Sec. 3.2, p.4)

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

- Why it matters: shows diffusion training $\approx$ denoising score matching across noise levels.

Eq. (13) — Discrete decoder for log-likelihood (Sec. 3.3, p.4–5)

(Per-dimension discretized Gaussian likelihood for integer pixels mapped to $[-1, 1]$.)

- Why it matters: enables discrete-data NLL reporting (Table 1, p.5).

Eq. (14) — Simplified training objective (Sec. 3.4, p.5)

$$L_{\text{simple}}(\theta) := \mathbb{E}_{t,x_0,\epsilon} \left[|\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t} x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon,, t)|^2 \right], \quad t \sim \text{Uniform}(1, \ldots, T).$$

- Why it matters: what they actually use for best sample quality (Table 1 "Ours (L_{simple})", p.5).

Eq. (15) — Predict x_0 from $\left(x_t, \epsilon_\theta\right)$ (Sec. 4.3, p.7)

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

- Why it matters: progressive decoding / compression viewpoint.

Eq. (16) — Alternate ELBO form (Sec. 4.3, p.7; App. A Eq. 26, p.14)

$$L = D_{KL}(q(x_T)|p(x_T)) + \mathbb{E}_q \left[\sum_{t \geq 1} D_{KL}(q(x_{t-1} \mid x_t)|p_\theta(x_{t-1} \mid x_t)) \right] + H(x_0).$$

- Why it matters: connects diffusion to autoregressive decoding when you reinterpret the “masking” diffusion.

Equation dependency map

- Eq. (1)+(2) → Eq. (3) (ELBO definition)
- Eq. (3) → Eq. (5) via App. A derivation (Eq. 17–22)
- Eq. (2) → Eq. (4) (closed-form marginal) → Eq. (6)–(7) (posterior)
- Eq. (5) + Eq. (6)–(7) + fixed σ_t^2 → Eq. (8)
- Eq. (4) reparameterization + Eq. (7) → Eq. (11) → Eq. (12)
- Eq. (12) (drop weights) → Eq. (14) training loss (Alg. 1)
- Eq. (11) + σ_t^2 → Alg. 2 sampling update
- Eq. (11) → Eq. (15) reconstruction estimate

Implementation translation

- Minimal set to code: Eq. 4 (forward sample), Eq. 14 (loss), Alg. 2 line 4 (reverse update).
- If you want true ELBO/NLL: implement Eq. 5 terms (Gaussian KL) + Eq. 13 decoder (discretized likelihood).
- Precompute and cache all scalar coefficients to avoid recomputing $\bar{\alpha}_t$ per step.

5) DERIVATION MAP (NO BIG JUMPS)

> Mini-explainer: **Reparameterization trick** turns sampling into deterministic function of noise: $x = \mu + \sigma\epsilon, \epsilon \sim \mathcal{N}(0,1)$.
> Tiny example: to sample $x \sim \mathcal{N}(3,4)$, do $x = 3 + 2\epsilon$.

Step-by-step arrow chain (assumptions → objective → final loop)

Step 1 (Sec. 2, Eq. 3, p.2–3): Start from ELBO-style objective

- Starting expression:

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

- Identity used: factorize p_θ and q by Markov property (Eq. 1–2).
- Resulting expression (paper jump → made explicit): see App. A Eq. (18).

- What changed and why: expanded the log of product into sum of logs so terms can be grouped per timestep.

Step 2 (App. A Eq. 17→18→19, p.14): Expand logs

- Starting expression (App. A Eq. 17):

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

- Identity used: $\log \frac{a \prod b_t}{\prod c_t} = \log a + \sum_t (\log b_t - \log c_t)$.
- Result (App. A Eq. 18–19):

$$L = \mathbb{E}_q \left[-\log p(x_T) - \sum_{t \geq 1} \log \frac{p_\theta(x_{t-1} \mid x_t)}{q(x_t \mid x_{t-1})} \right],$$

  then isolate $t = 1$ term to expose $p_\theta(x_0 \mid x_1)$.
- What changed: separated the decoder term ($t = 1$) from others to form L_0 later.

Step 3 (App. A Eq. 19→20, p.14): Insert Bayes for q (paper jump filled)

- Starting expression (App. A Eq. 19):

$$-\sum_{t>1} \log \frac{p_\theta(x_{t-1} \mid x_t)}{q(x_t \mid x_{t-1})}.$$

- Identity used (Bayes over the forward chain):

$$q(x_t \mid x_{t-1}) = \frac{q(x_{t-1} \mid x_t, x_0), q(x_t \mid x_0)}{q(x_{t-1} \mid x_0)}.$$

- Result (App. A Eq. 20): replace denominator accordingly.
- What changed and why: rewrites each term so $q(x_{t-1} \mid x_t, x_0)$ appears—this is Gaussian and tractable.

Step 4 (App. A Eq. 20→21→22, p.14): Turn log ratios into KLs

- Starting expression: (App. A Eq. 21)

$$\mathbb{E}_q \left[-\log \frac{p(x_T)}{q(x_T \mid x_0)} - \sum_{t>1} \log \frac{p_\theta(x_{t-1} \mid x_t)}{q(x_{t-1} \mid x_t, x_0)} - \log p_\theta(x_0 \mid x_1) \right].$$

- Identity used: $\mathbb{E}_q\left[\log \frac{q}{p}\right] = D_{KL}(q|p)$.
- Result (App. A Eq. 22) = main text Eq. (5): sum of KLs + decoder.
- What changed: recognized each expectation of a log ratio as a KL divergence.

Step 5 (Sec. 2, Eq. 6–7, p.3): Compute $q(x_{t-1} \mid x_t, x_0)$

- Starting expression: forward chain is linear Gaussian.
- Identity used: conditioning in jointly Gaussian variables; equivalently "multiply Gaussians" and complete the square.
- Result: Eq. (6)–(7) give $\tilde{\mu}_t, \tilde{\beta}_t$.
- What changed: replaced an intractable posterior with a closed-form Gaussian.

Step 6 (Sec. 3.2, Eq. 8, p.3–4): KL between Gaussians reduces to mean MSE

- Starting expression: $D_{KL}(\mathcal{N}(m_1, s_1^2 I)|\mathcal{N}(m_2, s_2^2 I))$.
- Identity used: closed-form Gaussian KL; if s_2^2 fixed and s_1^2 independent of θ, the only θ-dependent part is $|m_1 - m_2|^2/(2s_2^2)$.
- Result: Eq. (8).
- What changed: collapsed KL objective to regression on means when variances are fixed.

Step 7 (Sec. 3.2, Eq. 9–12, p.4): Reparameterize x_t and switch to predicting ϵ

- Starting expression: Eq. (8) mean loss in $\tilde{\mu}_t(x_t, x_0)$.
- Identity used: reparameterize Eq. (4) as $x_t(x_0, \epsilon) = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$ and substitute $x_0 = (x_t - \sqrt{1-\bar{\alpha}_t}\epsilon)/\sqrt{\bar{\alpha}_t}$.
- Result: Eq. (9) → Eq. (10) → Eq. (11) → Eq. (12) (details in next section).
- What changed: replaced "predict posterior mean" with "predict noise," yielding simpler learning signal.

Step 8 (Sec. 3.4, Eq. 14, p.5): Drop weight to get L_{simple}

- Starting expression: Eq. (12) weighted MSE.
- Identity used: none (design choice) — this is a **deliberate weighting change**.
- Result: Eq. (14).
- What changed: emphasizes larger-t denoising, improving sample quality (Sec. 4 discussion, p.5).

Implementation translation

- Derivation-to-code mapping:
  - Steps 1–4 matter only if you compute ELBO/NLL; otherwise skip.
  - Steps 7–8 are exactly what you implement for training: sample (x_0, t, ϵ), form x_t, MSE on ϵ.

- Numerical: computing Gaussian KLs for ELBO needs stable log-variance; but L_{simple} avoids it.

6) OBJECTIVE / LOSS (FINAL FORM + INTERPRETATION)

The "true" (bound-derived) per-timestep objective (Sec. 3.2, Eq. 12, p.4)

From Eq. (12):

$$\mathcal{L}_t(\theta) = \mathbb{E}_{x_0,\epsilon}\left[w_t \cdot |\epsilon - \epsilon_\theta(x_t, t)|^2\right], \quad x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon,$$

where

$$w_t = \frac{\beta_t^2}{2\sigma_t^2\alpha_t(1-\bar{\alpha}_t)}.$$

- Interpretation: each timestep trains denoising at a noise level; the weight w_t comes from the Gaussian KL geometry (Sec. 3.2, p.4).
- What it encourages: accurate noise prediction across all noise scales, with emphasis controlled by w_t.

> Mini-explainer: **Score** of a density is $\nabla_x \log p(x)$ (direction that increases likelihood). In Gaussian corruption models, denoising predicts something proportional to this gradient.
> Tiny example: for $p(x) = \mathcal{N}(0,1)$, score is $\nabla_x \log p(x) = -x$.

What they actually trained for best samples: L_{simple} (Sec. 3.4, Eq. 14, p.5)

$$L_{\text{simple}}(\theta) = \mathbb{E}_{t,x_0,\epsilon}\left[|\epsilon - \epsilon_\theta(x_t, t)|^2\right], \quad t \sim \text{Uniform}(1, \dots, T).$$

- Tradeoff: discards theoretically-derived w_t, making the objective a **weighted variational bound** (their words: a reweighting of terms) that empirically improves FID (Sec. 3.4–4, p.5–6).
- Practical meaning: focuses the network on harder denoising tasks (larger t) given their schedule; they claim this helps sample quality (Sec. 4 lead-in, p.5).

Decoder / likelihood term (Sec. 3.3, Eq. 13, p.4–5)

- If you want discrete-data NLL in bits/dim (Table 1, p.5), use their discretized Gaussian decoder:
  - For each pixel coordinate i, integrate a Gaussian pdf over the quantization bin $[\delta^-(x_0^i), \delta^+(x_0^i)]$ (Eq. 13).
- In L_{simple} training, they approximate the $t = 1$ term by "pdf $\times$ bin width," ignoring σ_1^2 and edge effects (Sec. 3.4, p.5).
  - **Ambiguity:** this approximation is only described verbally; exact implementation may differ (see "Ambiguities" below).

Implementation translation

- Training loss used: plain MSE between sampled ϵ and predicted ϵ_θ.
- If computing ELBO: implement weights w_t and also L_0 via discretized Gaussian (Eq. 13).
- Numerics: discretized Gaussian needs stable $\log(\Phi(b) - \Phi(a))$; use log-CDF tricks to avoid catastrophic cancellation for tiny bins.

7) ALGORITHMS (TRAINING + SAMPLING)

Pseudocode — Training (Alg. 1, p.4)

Inputs: dataset sampler for x_0, schedule arrays, model $\epsilon_\theta(\cdot, \cdot)$
Randomness: $t \sim \text{Uniform}(1, \dots, T)$, $\epsilon \sim \mathcal{N}(0, I)$

```
repeat
  x0 ← sample from data q(x0)
  t  ← Uniform({1,…,T})
  ε  ← Normal(0, I)
  xt ← sqrt(ᾱ_t) * x0 + sqrt(1−ᾱ_t) * ε        # Eq (4) reparam
  loss ← || ε − εθ(xt, t) ||^2                 # Eq (14)
  θ ← θ − η * ∇θ loss
until converged
```

(Alg. 1 is exactly this, using Eq. 14.)

Pseudocode — Sampling (Alg. 2, p.4)

Inputs: schedule arrays, σ_t choice, trained ϵ_θ
Randomness: initial $x_T \sim \mathcal{N}(0, I)$; per-step $z \sim \mathcal{N}(0, I)$ for $t > 1$

```
xT ← Normal(0, I)
for t = T, …, 1:
  if t > 1: z ← Normal(0, I) else z ← 0
  mean ← (1/sqrt(α_t)) ∗ ( x_t − (β_t/sqrt(1−ᾱ_t)) ∗ εθ(x_t,t) )   # Eq (11)
  x_{t−1} ← mean + σ_t ∗ z                                          # Alg 2 line 4
return x0
```

(Alg. 2 line 4 matches this form.)

> Mini-explainer: **Langevin dynamics** is a sampling procedure: take a step in the direction of the score (gradient of log-density) plus noise.
> Tiny example: $x \leftarrow x + \eta \nabla_x \log p(x) + \sqrt{2\eta}, z$; diffusion sampling looks like this with learned gradients (Sec. 3.2, p.4).

Minimal PyTorch-like skeleton (structure only)

```
# precompute betas[1..T], alphas, alpha_bars, sqrt_alpha_bars, sqrt_one_minus_alpha_bars

for step in range(num_steps):
    x0 = next(data_loader)                    # [B,C,H,W] in [−1,1]
    t  = randint(1, T, size=[B])              # int64
    eps = torch.randn_like(x0)
    xt  = sqrt_ab[t]*x0 + sqrt_1mab[t]*eps
    eps_pred = model(xt, t)                   # predicts eps, same shape
    loss = ((eps − eps_pred)**2).mean()
    loss.backward(); opt.step(); opt.zero_grad()
```

```
@torch.no_grad()
def sample(B):
    x = torch.randn([B,C,H,W])
    for t in range(T, 0, -1):
        z = torch.randn_like(x) if t > 1 else 0
        eps_pred = model(x, t)
        mean = inv_sqrt_a[t] * (x - (beta[t]/sqrt_1mab[t]) * eps_pred)
        x = mean + sigma[t]*z
    return x
```

Implementation translation

- Required tensors:
  - Scalars per t indexed by batch: `sqrt_ab[t]`, `sqrt_1mab[t]`, `inv_sqrt_a[t]`, `beta[t]`, `sigma[t]` → broadcast to image shape.
- Shapes: if `t` is `[B]`, gather scalars to `[B,1,1,1]` before multiply.
- Numerical: beware integer indexing off-by-one ($t \in [1, T]$); store arrays length $T+1$ with dummy at index 0.

8) DESIGN CHOICES & ABLATIONS

Choice	Options tried	What changed	Effect on results	My takeaway
Reverse mean parameterization	predict $\tilde{\mu}_t$ (baseline) vs predict ϵ (ours)	output head target and sampling formula	With variational bound + fixed variance, both okay; with MSE-style objective, ϵ prediction wins big (Table 2, p.5).	Predicting ϵ makes training robust under simplified loss.
Training objective	true bound L vs L_{simple}	weighted KL-derived vs unweighted MSE	L_{simple} gives best FID on CIFAR10 (Table 1–2, p.5).	Empirically, reweighting towards larger noise scales helps samples.

Choice	Options tried	What changed	Effect on results	My takeaway
Reverse variance Σ_θ	learned diagonal vs fixed isotropic	predict log-variance vs constant $\sigma_t^2 I$	learning variance unstable / poorer sample quality (Sec. 4.2, p.6; Table 2, p.5).	Keep variance fixed for stability (in this paper's setup).
Fixed variance schedule	$\sigma_t^2 = \beta_t$ vs $\sigma_t^2 = \tilde{\beta}_t$	sampling noise scale	Similar results (Sec. 3.2, p.3).	Either works; choose one and stick to consistent derivation.
β_t schedule	constant, linear, quadratic (constrained so $L_T \approx 0$)	noise injection over time	They picked linear $\beta_1 = 10^{-4}$ to $\beta_T = 0.02$ (App. B, p.15).	Ensure near-complete destruction of signal at T to avoid prior mismatch.
Architecture	U-Net / PixelCNN++-like with GroupNorm + attention	model capacity and inductive bias	Enables strong image results (Sec. 4, p.5; App. B, p.15).	U-Net + attention at 16×16 is a strong default.
Regularization	dropout on CIFAR10	prevent overfitting artifacts	dropout 0.1 improved CIFAR samples; set to 0 elsewhere (App. B, p.15).	CIFAR benefits from dropout; larger datasets maybe not.

Implementation translation

- Ablation-sensitive knobs: objective (weighted vs unweighted), target type (ϵ vs $\tilde{\mu}$), variance fixed vs learned.
- If reproducing Table 2: you must implement multiple heads/parameterizations; easiest is always predict ϵ and derive μ_θ via Eq. 11.

9) IMPLEMENTATION & REPRODUCTION NOTES

Datasets + preprocessing

- Data: CIFAR10, LSUN (Bedroom/Church/Cat), CelebA-HQ. Loaded via TFDS for CIFAR10/CelebA-HQ; LSUN prepared using StyleGAN code (App. B, p.15).
- Scaling: integer pixels $0, \ldots, 255$ scaled linearly to $[-1, 1]$ (Sec. 3.3, p.4).
- Augmentation: random horizontal flips for CIFAR10 and all others except LSUN Bedroom (App. B, p.15).

Model architecture

- Backbone: U-Net similar to PixelCNN++ / Wide ResNet; GroupNorm throughout (Sec. 4, p.5; App. B, p.15).
- Resolutions:
  - 32×32 models: 4 resolutions ($32 \to 16 \to 8 \to 4$).
  - 256×256 models: 6 resolutions.
- Blocks: 2 convolutional residual blocks per resolution; self-attention blocks at 16×16 between conv blocks (App. B, p.15).
- Time conditioning: add Transformer sinusoidal position embedding of t into each residual block (App. B, p.15).
- Params: CIFAR model 35.7M; LSUN/CelebA-HQ 114M; larger LSUN Bedroom 256M by increasing filters (App. B, p.15).

Optimization + training

- Hardware: TPU v3-8 (≈ 8 V100 GPUs) (App. B, p.15).
- Steps/speeds: CIFAR 21 steps/sec, batch 128, 800k steps (~10.6h); sampling 256 images ~17s. 256^2 models: 2.2 steps/sec, batch 64, sampling 128 images ~300s (App. B, p.15).
- Training steps: CelebA-HQ 0.5M; LSUN Bedroom 2.4M; LSUN Cat 1.8M; LSUN Church 1.2M; large Bedroom 1.15M (App. B, p.15).
- Optimizer: Adam with “standard values” (App. B, p.15).
- LR: 2×10^{-4} (CIFAR); 2×10^{-5} for 256×256 (stability) (App. B, p.15).
- EMA: decay 0.9999 (App. B, p.15).
- Dropout: CIFAR 0.1; others 0 (App. B, p.15).

Evaluation details

- CIFAR metrics on 50k samples; Inception from OpenAI code; FID from TTUR code; LSUN FID from StyleGAN2 repo (App. B, p.15).
- FID computed vs training set (standard); test-set FID reported as 5.24 as a check (Sec. 4.1, p.5–6).

Gotchas & stability notes

1. Off-by-one timestep indexing ($t \in [1, T]$; special-case $t = 1$ with $z = 0$ in sampling). (Alg. 2, p.4)
2. Store $\bar{\alpha}_t$ in float64; cumulative products in float32 can drift for $T = 1000$.
3. If you try learning Σ_θ, expect instability (Table 2, p.5).
4. Data must be scaled to $[-1, 1]$ for the schedule they used (Sec. 3.3; Sec. 4, p.4–5).
5. Discretized decoder (Eq. 13) needs careful numerical log-CDF if you compute NLL.

Hyperparameters that matter most (ranked)

1. β_t schedule endpoints + shape (App. B, p.15).
2. Objective choice (L_{simple} vs weighted) (Eq. 14 vs Eq. 12; Table 1–2, p.5).
3. Model capacity + attention placement (App. B, p.15).
4. Dropout on CIFAR10 (App. B, p.15).
5. EMA decay (App. B, p.15).

Implementation translation

- Repro checklist for code:
  - Match data scaling and schedule exactly.
  - Use sinusoidal embedding for t injected into every residual block.
  - Evaluate with EMA weights (likely critical though not quantified).
- Inference (label as inference): “standard Adam values” likely means $(\beta_1, \beta_2, \epsilon) = (0.9, 0.999, 10^{-8})$; paper doesn’t explicitly print them.

10) RESULTS & EVALUATION

Metrics (direction)

- IS (Inception Score): higher is better.

- FID: lower is better.
- NLL in bits/dim: lower is better (Table 1, p.5).

Main tables (best numbers)

- CIFAR10 (unconditional): "Ours (L_{simple})" achieves IS 9.46 ± 0.11, FID 3.17, NLL ≤ 3.75 (Table 1, p.5).
- LSUN 256×256: FID Bedroom 6.36 (and 4.90 for "large"), Church 7.89, Cat 19.75 (Table 3, p.13).

Baselines & fairness

- They compare to GANs, autoregressive, and score-based models in Table 1; compute/data parity varies across papers (not fully controlled). Table itself is a literature compilation, so treat as "best-effort."
- They note FID computed vs training set as standard; also report test-set FID 5.24 (Sec. 4.1, p.5–6).

Failure modes / weaknesses

- Likelihood: their bits/dim are not competitive with top likelihood-based models like Sparse Transformer (discussion Sec. 4.3, p.6–7).
- Sampling cost: $T = 1000$ reverse steps; 256×256 sampling is slow (~300s per batch of 128 on TPU v3-8) (App. B, p.15).

How likelihood is computed

- Through the variational bound with discretized decoder (Eq. 5 + Eq. 13) reporting bits/dim (Table 1).

Implementation translation

- To reproduce reported NLL: implement full ELBO terms (Eq. 5) and discretized decoder (Eq. 13).
- To reproduce reported FID/IS: generate 50k samples using EMA weights and same metric code references (App. B, p.15).

11) INTUITION & CONNECTIONS

Mechanistic intuition (not vibes)

- Forward diffusion slowly destroys information; at time t, x_t retains a fraction $\sqrt{\bar{\alpha}_t}$ of x_0 signal and adds Gaussian noise $\sqrt{1-\bar{\alpha}_t}\epsilon$ (Eq. 4, Sec. 2, p.2).
- The optimal reverse step should "subtract the right amount of noise" conditioned on x_t. Because the chain is Gaussian, the posterior mean $\tilde{\mu}_t(x_t, x_0)$ is linear in (x_t, x_0) (Eq. 7, Sec. 2, p.3).
- But at sampling time you don't know x_0, so you learn a network that predicts the latent noise ϵ used to form x_t; from ϵ you can compute an estimate of x_0 (Eq. 15, Sec. 4.3, p.7) and hence the reverse mean (Eq. 11, Sec. 3.2, p.4).
- Training is stable because, with fixed variances, each KL term is (up to a constant) just a squared error between Gaussian means (Eq. 8, Sec. 3.2, p.3–4).

Related ideas/papers (connections the paper emphasizes)

- Sohl-Dickstein et al. (original diffusion): same ELBO decomposition idea; App. A derivation is credited to them (App. A, p.14).
- Denoising score matching + annealed Langevin dynamics: Eq. (12) resembles multi-noise denoising score matching, and Alg. 2 resembles Langevin sampling (Sec. 3.2, p.4; Related Work p.8–9).
- Autoregressive decoding view: alternate ELBO form (Eq. 16, Sec. 4.3, p.7) shows diffusion can mimic AR factorization under a masking-style diffusion.
- Energy-based models: via known "score matching ↔ EBM" connection (Related Work, p.8–9).

What this paper secretly is

A **variationally-trained, Langevin-like sampler** where the learned network provides the denoising/score signal across noise levels; L_{simple} is a practical reweighting that improves perceptual sample quality.

Implementation translation

- If you think "score model": ϵ_θ is the primitive; everything else (reverse mean, x_0 estimate) is derived.
- If you think "VAE": $x_{1:T}$ are latents; ELBO decomposes across timesteps; but training can skip explicit KLs using the ϵ-MSE surrogate.

12) LIMITATIONS, ASSUMPTIONS, AND OPEN QUESTIONS

Explicit limitations (from paper)

- Progressive compression is a proof-of-concept and not practical due to needing minimal random coding (discussion near Table 4 / progressive compression, p.13).
- Sampling is slow because reverse chain length is $T = 1000$ (Sec. 4, p.5; App. B timing p.15).

Implicit limitations (inferred; labeled)

- **Inference:** fixed isotropic variance may limit likelihood performance or calibration; learning variance was unstable in their setup, but later work might stabilize it. (Based on Table 2 instability + their choice.)
- **Inference:** their "ignore small-t" effect is schedule-dependent; different β_t schedules could change what L_{simple} emphasizes.

Open questions / what I'd test next (5–10)

1. Can we reduce T drastically while keeping FID (faster sampling) without changing the training objective?
2. What exact weighting w_t (between Eq. 12 and Eq. 14) optimizes a given perceptual metric?
3. Can we stabilize learned variances (Table 2 suggests no) using better parameterization/regularization?
4. How sensitive is performance to the "$L_T \approx 0$" constraint on schedules?
5. Replace discrete decoder (Eq. 13) with a stronger conditional decoder—does NLL improve without hurting samples?
6. Architecture: move/add attention resolutions beyond 16×16—how does it scale?
7. Objective: predict x_0 instead of ϵ (they say it was worse early on)—under what conditions does it become better?
8. Data modalities beyond images: does the same schedule/architecture trick transfer?

What breaks if assumptions are violated?

- If β_t not "small" and $q(x_T \mid x_0)$ not close to $\mathcal{N}(0, I)$, then prior mismatch can cause distribution shift during sampling (Appendix C point 3, p.15).
- If you omit the forward scaling by $\sqrt{1-\beta_t}$ (like some score models), inputs may not stay consistently scaled and sampling can degrade (Appendix C point 2, p.15).

Implementation translation

- The most fragile parts: schedule endpoints, ensuring x_T is near standard normal, and consistent data scaling to $[-1, 1]$.
- If you deviate: re-check $\mathrm{SNR}(t) = \bar{\alpha}_t/(1-\bar{\alpha}_t)$ and confirm L_T is tiny.

13) "STEAL THIS" SECTION (PORTABLE IDEAS)

1. **ϵ-prediction parameterization** (Eq. 11, Sec. 3.2, p.4)
   - Why it helps: converts reverse-step learning into plain denoising regression; simplifies sampling update (Alg. 2).
   - Apply elsewhere: anytime you have $x = \mathrm{signal} + \mathrm{noise}$, predict the noise to stabilize training.
2. **One-shot noising via closed-form $q(x_t \mid x_0)$** (Eq. 4, Sec. 2, p.2)
   - Why it helps: $O(1)$ training-time corruption for random t (no forward loop).
3. **ELBO → sum of local KLs (variance reduction)** (Eq. 5, Sec. 2, p.3; App. A Eq. 17–22, p.14)
   - Why it helps: replaces high-variance Monte Carlo over entire latent chain with tractable Gaussian KLs.
4. **Objective reweighting for perceptual quality** (Eq. 12 → Eq. 14, Sec. 3.4, p.5)
   - Why it helps: shifts learning focus toward harder/noisier denoising tasks; improves FID.
5. **Time conditioning via sinusoidal embeddings injected everywhere** (App. B, p.15)
   - Why it helps: shares parameters across timesteps; makes one network handle all noise levels.

Implementation translation

- These ideas are modular: you can swap in any backbone that maps $(x_t, t) \mapsto \epsilon$.
- The "must keep" invariants: correct scalar coefficients from the schedule; correct handling of t conditioning.

14) SELF-TEST (FOR LEARNING)

10 short questions

1. What does β_t control?

2. Define α_t and $\bar{\alpha}_t$.
3. Write the closed-form for $q(x_t \mid x_0)$.
4. What is the training target in L_{simple}?
5. In Alg. 2, why is $z = 0$ when $t = 1$?
6. What does Eq. (5) decompose L into?
7. What is $\tilde{\mu}_t(x_t, x_0)$ used for?
8. Why does KL between Gaussians become an MSE when variances are fixed?
9. How do you estimate x_0 from (x_t, ϵ_θ)?
10. Where is self-attention used in their architecture?

▶ **Answers (short)**

5 medium questions

1. Derive Eq. (8) from the Gaussian KL formula (show what becomes constant).
2. Starting from Eq. (7) and $x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon$, derive the simplified posterior mean form used in Eq. (11).
3. Explain why dropping the weight in Eq. (12) changes what the model emphasizes across timesteps.
4. Why does ensuring $L_T \approx 0$ matter for sampling?
5. Compare $\sigma_t^2 = \beta_t$ vs $\sigma_t^2 = \tilde{\beta}_t$: what intuition does the paper give?

▶ **Answers (medium)**

2 hard questions

1. Suppose you want $10\times$ faster sampling. What modifications to the method would you try first, and what failure mode would you watch for?
2. If you replace the linear β_t schedule with a much steeper schedule (larger β_t early), predict how it would affect (a) L_T, (b) training difficulty, and (c) sample quality.

▶ **Answers (hard; reasoning)**

Implementation translation

- Use these questions to sanity-check your code: if changing T or schedule breaks sampling, suspect coefficient math or L_T mismatch.
- A practical diagnostic: compute empirical $\mathrm{SNR}(t) = \bar{\alpha}_t/(1 - \bar{\alpha}_t)$ curve and ensure it's smooth and ends very low.

15) FINAL CHECKLIST (MUST INCLUDE)

- [x] I listed the canonical equations and explained each
- [x] I made a derivation map with no big jumps (and labeled “paper jump” spots)
- [x] I gave sampling + training pseudocode (plus minimal PyTorch-like skeleton)
- [x] I extracted all experimental/repro details (datasets, schedules, arch, optimization, compute)
- [x] I summarized results + ablations + limitations
- [x] I wrote a complete notation glossary with shapes
- [x] I included “gotchas” and “what to test next”