Generative models for images VI: Diffusion models

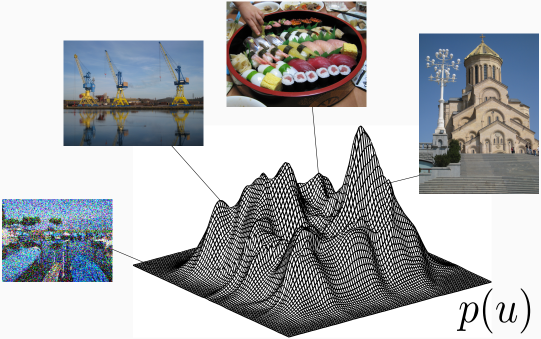
Bruno Galerne
bruno.galerne@univ-orleans.fr

Master MVA 2024-25
Tuesday February 18, 2024

Institut Denis Poisson
Université d'Orléans, Université de Tours, CNRS
Institut universitaire de France (IUF)

Generative models

1. Model and/or learn a distribution $p(u)$ on the space of images.



(source: Charles Deledalle)

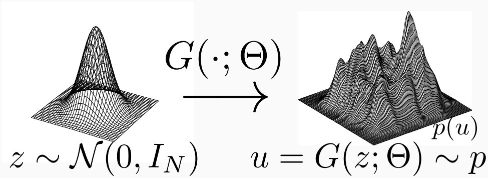
The images may represent:

- different instances of the same texture image,
- all images naturally described by a dataset of images,
- any image

2. Generate samples from this distribution.

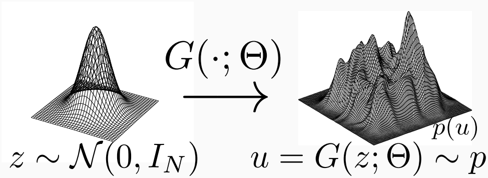
Generative models

1. Model and/or learn a distribution $p(u)$ on the space of images.
2. Generate samples from this distribution.



- z is a generic source of randomness, often called the latent variable.
- If $G(\cdot;\Theta)$ is known, then $p = G(\cdot;\Theta)_{\#}\mathcal{N}(0, I_n)$ is the push-forward of the latent distribution.

1. Model and/or learn a distribution $p(u)$ on the space of images.
2. Generate samples from this distribution.



- z is a generic source of randomness, often called the latent variable.
- If $G(\cdot;\Theta)$ is known, then $p = G(\cdot;\Theta)_{\#}\mathcal{N}(0, I_n)$ is the push-forward of the latent distribution.

The generator $G(\cdot;\Theta)$ can be:

- A deterministic function (e.g. convolution operator),
- A neural network with learned parameter,
- An iterative optimization algorithm (gradient descent,...),
- A stochastic sampling algorithm (e.g. MCMC, Langevin diffusion,...).

Basics on diffusion models

Adding noise to images

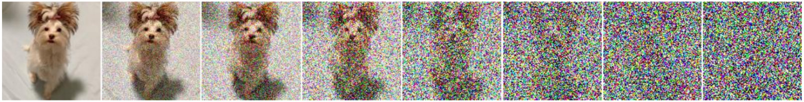
- We are given an input dataset

$$\mathcal{D} = \{\boldsymbol{x}^{(i)},\ i = 1, \dots, N\} \subset \mathbb{R}^d$$

- We assume that these images are independent samples of a common distribution p_0 over $\mathbb{R}^d$.
- Consider the random process that consists of adding noise to images:

$$\boldsymbol{x}_t = \boldsymbol{x}_0 + \boldsymbol{w}_t, \quad t \in [0, T]$$

where $\boldsymbol{x}_0 \sim p_0$ is a sample image and $\boldsymbol{w}_t$ is a Brownian motion (also called Wiener process).



(source: (Song et al., 2021b))

Real-valued: A standard (real-valued) **Brownian motion** (also called **Wiener process** is a stochastic process $(w_t)_{t\geq 0}$ such that

- $w_0 = 0$.
- With probability one, the function $t \mapsto w_t$ is continuous.
- The process $(w_t)_{t\geq 0}$ has stationary independent increments.
- $w_t \sim \mathcal{N}(0, t)$.

Direct consequences:

- For $s < t$, w_s and $w_t - w_s$ are independent and $w_{t-s} \sim \mathcal{N}(0, t-s)$.
- Markovian random field.

$\mathbb{R}^d$**-valued:** A standard $\mathbb{R}^d$-valued Brownian motion $(\boldsymbol{w}_t)_{t\geq 0}$ is made of d independent real-valued Brownian motions

$$\boldsymbol{w}_t = (w_{t,1}, \ldots, w_{t,d}) \in \mathbb{R}^d.$$

Ito integral on $[0, T]$:

Given a process $(\boldsymbol{x}_t)_{t\in[0,T]}$ adapted to the filtration $\mathcal{F}_t = \sigma(\boldsymbol{w}_s, s \leq t)$, one defines

$$\int_0^t \boldsymbol{x}_s d\boldsymbol{w}_s \quad \text{as the } L^2 \text{ limit of} \quad \sum_{j=0}^{k-1} \boldsymbol{x}_{t_j} \odot (\boldsymbol{w}_{t_{j+1}} - \boldsymbol{w}_{t_j})$$

when the minimal step of the partition $0 \leq t_0 \leq \cdots \leq t_k \leq T$ tends to 0.

- In particular, for a deterministic function $s \mapsto g(s)$, $\int_0^t g(s) d\boldsymbol{w}_s$ is a normal variable with mean 0 and variance $\sigma^2 = \int_0^t g^2(s) ds$.

Adding noise to images

- Adding noise to images: $\boldsymbol{x}_t = \boldsymbol{x}_0 + \boldsymbol{w}_t, \quad t \in [0, T]$.
- This corresponds to the stochastic differential equation (SDE):

$$d\boldsymbol{x}_t = d\boldsymbol{w}_t \quad \text{with initial condition } \boldsymbol{x}_0 \sim p_0.$$

- We denote by p_t the distribution of $\boldsymbol{x}_t$ at time $t \in [0, T]$. What is p_t?

$$p_t = p_0 * \mathcal{N}(\mathbf{0}, tI_d)$$

- This corresponds to applying the heat equation starting from p_0:

$$\partial_t p_t(\boldsymbol{x}) = \frac{1}{2}\Delta_{\boldsymbol{x}} p_t(\boldsymbol{x}) \quad \text{with } p_{t=0} = p_0.$$

  This PDE is called the **Fokker-Planck equation** associated with the SDE.
- This is an example of diffusion equation.

Diffusion SDE and Fokker-Planck equation

- More generally we will consider diffusion SDE of the form (Song et al., 2021b):

$$d\boldsymbol{x}_t = \boldsymbol{f}(\boldsymbol{x}_t, t)dt + g(t)d\boldsymbol{w}_t$$

  where
  - $\boldsymbol{f} : \mathbb{R}^d \times [0, T] \to \mathbb{R}^d$ is called the **drift**: External deterministic force that drives $\boldsymbol{x}_t$ in the direction $\boldsymbol{f}(\boldsymbol{x}_t, t)$,
  - $g : [0, T] \to [0, +\infty)$ is the **diffusion coefficient**.

- The corresponding Fokker-Planck equation is

$$\partial_t p_t(\boldsymbol{x}) = -\operatorname{div}_{\boldsymbol{x}} \left(\boldsymbol{f}(\boldsymbol{x}, t) p_t(\boldsymbol{x})\right) + \frac{1}{2} g(t)^2 \Delta_{\boldsymbol{x}} p_t(\boldsymbol{x})$$

  that is,

$$\partial_t p_t(\boldsymbol{x}) = -\sum_{k=1}^{d} \partial_{\boldsymbol{x}_k} \left[\boldsymbol{f}_k(\boldsymbol{x}, t) p_t(\boldsymbol{x})\right] + \frac{1}{2} g(t)^2 \sum_{k=1}^{d} \partial^2_{\boldsymbol{x}_k} p_t(\boldsymbol{x}).$$

Diffusion SDE: Two examples

$$d\boldsymbol{x}_t = \boldsymbol{f}(\boldsymbol{x}_t, t)dt + g(t)d\boldsymbol{w}_t$$

Example 1: Variance exploding diffusion (VE-SDE)

SDE: $d\boldsymbol{x}_t = d\boldsymbol{w}_t$
Solution: $\boldsymbol{x}_t = \boldsymbol{x}_0 + \boldsymbol{w}_t$
Variance: $\mathrm{Var}(\boldsymbol{x}_t) = \mathrm{Var}(\boldsymbol{x}_0) + t$

Example 2: Variance preserving diffusion (VP-SDE)

SDE: $d\boldsymbol{x}_t = -\beta_t \boldsymbol{x}_t dt + \sqrt{2\beta_t} d\boldsymbol{w}_t$
Solution: $\boldsymbol{x}_t = e^{-B_t}\boldsymbol{x}_0 + \int_0^t e^{B_s - B_t}\sqrt{2\beta_s} d\boldsymbol{w}_s$ with $B_t = \int_0^t \beta_s ds$
Variance: $\mathrm{Var}(\boldsymbol{x}_t) = e^{-2B_t}\mathrm{Var}(\boldsymbol{x}_0) + 1 - e^{-2B_t}$ $= 1$ if $\mathrm{Var}(\boldsymbol{x}_0) = 1.$

Both variants have the form $\boldsymbol{x}_t = a_t \boldsymbol{x}_0 + b_t \boldsymbol{Z}_t$: $\boldsymbol{x}_t$ is a rescaled noisy version of $\boldsymbol{x}_0$ and the noise is more and more predominant as time grows.

$$d\boldsymbol{x}_t = \boldsymbol{f}(\boldsymbol{x}_t, t)dt + g(t)d\boldsymbol{w}_t$$

In general we do not have a close form formula for $\boldsymbol{x}_t$.

Diffusion SDEs can be approximately simulated using numerical schemes such as the **Euler-Maruyama sheme**:

- Using the time step $h = T/N$ with $N+1$ times $t_n = nh$, $n \in \{0, \dots N\}$, define $\boldsymbol{X}_0 = \boldsymbol{x}_0$ and

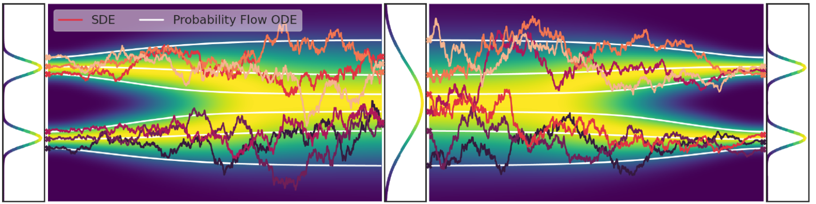
$$\boldsymbol{X}_{n+1} = \boldsymbol{X}_n + \boldsymbol{f}(\boldsymbol{X}_n, t_n)h + g(t_n)\left(\boldsymbol{w}_{t_{n+1}} - \boldsymbol{w}_{t_n}\right), \quad n = 1, \dots, N-1.$$

- Remark that $\boldsymbol{w}_{t_{n+1}} - \boldsymbol{w}_{t_n} \sim \mathcal{N}(\boldsymbol{0}, hI_d)$ and is independent of $\boldsymbol{X}_n$:

$$\boldsymbol{X}_{n+1} = \boldsymbol{X}_n + \boldsymbol{f}(\boldsymbol{X}_n, t_n)h + g(t_n)\sqrt{h}\boldsymbol{Z}_n, \quad \text{with } \boldsymbol{Z}_n \sim \mathcal{N}(\boldsymbol{0}, I_d), \quad n = 1, \dots, N-1.$$

Reversed diffusion

- For diffusion SDEs, as t grows p_t is closer and closer to a normal distribution.
- We will consider that at the final time $t = T$ large enough so that p_T can be considered to be a normal distribution.
- For generative modeling, we want to reverse the process:
  - Start by generating $\boldsymbol{x}_T \sim p_T \approx \mathcal{N}(\mathbf{0}, \sigma_T^2 I_d)$.
  - Simulate $(\boldsymbol{x}_{T-t})_{t\in[0,T]}$ such that $\boldsymbol{x}_{T-t} \sim p_{T-t}$.



(source: (Song and Ermon, 2020))

Reversed diffusion

Reversed diffusion: What is the SDE satisfied by $\boldsymbol{x}_{T-t}$?

$$d\boldsymbol{x}_t = \boldsymbol{f}(\boldsymbol{x}_t, t)dt + g(t)d\boldsymbol{w}_t$$

has the associated Fokker-Planck equation

$$\partial_t p_t(\boldsymbol{x}) = -\operatorname{div}_{\boldsymbol{x}}\left(\boldsymbol{f}(\boldsymbol{x}, t)p_t(\boldsymbol{x})\right) + \frac{1}{2}g(t)^2\Delta_{\boldsymbol{x}} p_t(\boldsymbol{x}).$$

Let us derive the Fokker-Planck equation for $q_t = p_{T-t}$ the distribution function of $\boldsymbol{y}_t = \boldsymbol{x}_{T-t}$.

$$\begin{aligned}
\partial_t q_t(\boldsymbol{x}) &= -\partial_t p_{T-t}(\boldsymbol{x}) \\
&= \operatorname{div}_{\boldsymbol{x}}\left(\boldsymbol{f}(\boldsymbol{x}, T-t)p_{T-t}(\boldsymbol{x})\right) - \frac{1}{2}g(T-t)^2\Delta_{\boldsymbol{x}} p_{T-t}(\boldsymbol{x}) \\
&= \operatorname{div}_{\boldsymbol{x}}\left(\boldsymbol{f}(\boldsymbol{x}, T-t)q_t(\boldsymbol{x})\right) - \frac{1}{2}g(T-t)^2\Delta_{\boldsymbol{x}} q_t(\boldsymbol{x}) \\
&= \operatorname{div}_{\boldsymbol{x}}\left(\boldsymbol{f}(\boldsymbol{x}, T-t)q_t(\boldsymbol{x})\right) + \left(-1 + \frac{1}{2}\right)g(T-t)^2\Delta_{\boldsymbol{x}} q_t(\boldsymbol{x})
\end{aligned}$$

Reversed diffusion

$$
\begin{aligned}
&\partial_t q_t(\boldsymbol{x}) \\
&= \operatorname{div}_{\boldsymbol{x}} \left(\boldsymbol{f}(\boldsymbol{x}, T-t) q_t(\boldsymbol{x})\right) + \left(-1 + \frac{1}{2}\right) g(T-t)^2 \Delta_{\boldsymbol{x}} q_t(\boldsymbol{x}) \\
&= \operatorname{div}_{\boldsymbol{x}} \left(\boldsymbol{f}(\boldsymbol{x}, T-t) q_t(\boldsymbol{x}) - g(T-t)^2 \nabla_{\boldsymbol{x}} q_t(\boldsymbol{x})\right) + \frac{1}{2} g(T-t)^2 \Delta_{\boldsymbol{x}} q_t(\boldsymbol{x}) \\
&= \operatorname{div}_{\boldsymbol{x}} \left(\left[\boldsymbol{f}(\boldsymbol{x}, T-t) - g(T-t)^2 \frac{\nabla_{\boldsymbol{x}} q_t(\boldsymbol{x})}{q_t(\boldsymbol{x})}\right] q_t(\boldsymbol{x})\right) + \frac{1}{2} g(T-t)^2 \Delta_{\boldsymbol{x}} q_t(\boldsymbol{x}) \\
&= -\operatorname{div}_{\boldsymbol{x}} \left(\left[-\boldsymbol{f}(\boldsymbol{x}, T-t) + g(T-t)^2 \nabla_{\boldsymbol{x}} \log q_t(\boldsymbol{x})\right] q_t(\boldsymbol{x})\right) + \frac{1}{2} g(T-t)^2 \Delta_{\boldsymbol{x}} q_t(\boldsymbol{x})
\end{aligned}
$$

This is the Fokker-Planck equation associated with the diffusion SDE:

$$
d\boldsymbol{y}_t = \left[-\boldsymbol{f}(\boldsymbol{y}_t, T-t) + g(T-t)^2 \nabla_{\boldsymbol{x}} \log p_{T-t}(\boldsymbol{y}_t)\right] dt + g(T-t) d\boldsymbol{w}_t.
$$

Reversed diffusion

Forward diffusion:

$$d\boldsymbol{x}_t = \boldsymbol{f}(\boldsymbol{x}_t, t)dt + g(t)d\boldsymbol{w}_t$$

Backward diffusion: $\boldsymbol{y}_t = \boldsymbol{x}_{T-t}$

$$d\boldsymbol{y}_t = \left[-\boldsymbol{f}(\boldsymbol{y}_t, T-t) + g(T-t)^2 \nabla_{\boldsymbol{x}} \log p_{T-t}(\boldsymbol{y}_t)\right] dt + g(T-t)d\boldsymbol{w}_t.$$

- Same diffusion coefficient.
- Opposite drift term with additional distribution correction:

  $$g(T-t)^2 \nabla_{\boldsymbol{x}} \log p_{T-t}(\boldsymbol{y}_t)$$

  drives the diffusion in regions with high p_{T-t} probability.
- $\boldsymbol{x} \mapsto \nabla_{\boldsymbol{x}} \log p_t(\boldsymbol{x})$ is called the (Stein) **score** of the distribution.
- Rigorous results from SDE litterature ((Anderson, 1982) (Haussmann and Pardoux, 1986)) (measurability issues, the filtration is also reversed...).

Reversed diffusion

Forward diffusion:

$$d\boldsymbol{x}_t = \boldsymbol{f}(\boldsymbol{x}_t, t)dt + g(t)d\boldsymbol{w}_t$$

Backward diffusion: $\boldsymbol{y}_t = \boldsymbol{x}_{T-t}$

$$d\boldsymbol{y}_t = \left[-\boldsymbol{f}(\boldsymbol{y}_t, T-t) + g(T-t)^2 \nabla_{\boldsymbol{x}} \log p_{T-t}(\boldsymbol{y}_t)\right] dt + g(T-t)d\boldsymbol{w}_t.$$

- Same diffusion coefficient.
- Opposite drift term with additional distribution correction:

$$g(T-t)^2 \nabla_{\boldsymbol{x}} \log p_{T-t}(\boldsymbol{y}_t)$$

  drives the diffusion in regions with high p_{T-t} probability.
- $\boldsymbol{x} \mapsto \nabla_{\boldsymbol{x}} \log p_t(\boldsymbol{x})$ is called the (Stein) **score** of the distribution.
- Rigorous results from SDE litterature ((Anderson, 1982) (Haussmann and Pardoux, 1986)) (measurability issues, the filtration is also reversed...).
- Can we simulate this backward diffusion using Euler-Maruyama ?

$$\boldsymbol{X}_{n+1} = \boldsymbol{X}_n + \boldsymbol{f}(\boldsymbol{X}_n, t_n)h + g(t)\sqrt{h}\boldsymbol{Z}_n, \quad \text{with } Z_n \sim \mathcal{N}(\boldsymbol{0}, I_d), \quad n = 1, \dots, N-1.$$

Learning the score function: Denoising score matching

- **Goal:** Estimate the score $\boldsymbol{x} \mapsto \nabla_{\boldsymbol{x}} \log p_t(\boldsymbol{x})$ using only available samples $(\boldsymbol{x}_0, \boldsymbol{x}_t)$.
- For the models of interests, $\boldsymbol{x}_t = a_t \boldsymbol{x}_0 + b_t \boldsymbol{Z}_t$ is a rescaled noisy version of $\boldsymbol{x}_0$ (both a_t and b_t have known analytical expressions).
- Explicit conditional distribution: $p_{t|0}(\boldsymbol{x}_t|\boldsymbol{x}_0) = \mathcal{N}(a_t\boldsymbol{x}_0, b_t^2 I_d)$.

$$
\begin{aligned}
p_t(\boldsymbol{x}_t) &= \int_{\mathbb{R}^d} p_{0,t}(\boldsymbol{x}_0, \boldsymbol{x}_t) d\boldsymbol{x}_0 = \int_{\mathbb{R}^d} p_{t|0}(\boldsymbol{x}_t|\boldsymbol{x}_0) p_0(\boldsymbol{x}_0) d\boldsymbol{x}_0 \\
\nabla_{\boldsymbol{x}_t} p_t(\boldsymbol{x}_t) &= \int_{\mathbb{R}^d} \nabla_{\boldsymbol{x}_t} p_{t|0}(\boldsymbol{x}_t|\boldsymbol{x}_0) p_0(\boldsymbol{x}_0) d\boldsymbol{x}_0 \\
\nabla_{\boldsymbol{x}_t} \log p_t(\boldsymbol{x}_t) = \frac{\nabla_{\boldsymbol{x}_t} p_t(\boldsymbol{x}_t)}{p_t(\boldsymbol{x}_t)} &= \int_{\mathbb{R}^d} \nabla_{\boldsymbol{x}_t} p_{t|0}(\boldsymbol{x}_t|\boldsymbol{x}_0) \frac{p_0(\boldsymbol{x}_0)}{p_t(\boldsymbol{x}_t)} d\boldsymbol{x}_0 \\
&= \int_{\mathbb{R}^d} \left[\nabla_{\boldsymbol{x}_t} \log p_{t|0}(\boldsymbol{x}_t|\boldsymbol{x}_0)\right] p_{t|0}(\boldsymbol{x}_t|\boldsymbol{x}_0) \frac{p_0(\boldsymbol{x}_0)}{p_t(\boldsymbol{x}_t)} d\boldsymbol{x}_0 \\
&= \int_{\mathbb{R}^d} \left[\nabla_{\boldsymbol{x}_t} \log p_{t|0}(\boldsymbol{x}_t|\boldsymbol{x}_0)\right] p_{0|t}(\boldsymbol{x}_0|\boldsymbol{x}_t) d\boldsymbol{x}_0
\end{aligned}
$$

Conclusion:

$$
\nabla_{\boldsymbol{x}_t} \log p_t(\boldsymbol{x}_t) = \mathbb{E}_{\boldsymbol{x}_0 \sim p_{0|t}(\boldsymbol{x}_0|\boldsymbol{x}_t)} \left[\nabla_{\boldsymbol{x}_t} \log p_{t|0}(\boldsymbol{x}_t|\boldsymbol{x}_0)\right] = \mathbb{E}\left[\nabla_{\boldsymbol{x}_t} \log p_{t|0}(\boldsymbol{x}_t|\boldsymbol{x}_0) | \boldsymbol{x}_t\right]
$$

Learning the score function: Denoising score matching

$$\nabla_{\boldsymbol{x}_t} \log p_t(\boldsymbol{x}_t) = \mathbb{E}_{\boldsymbol{x}_0 \sim p_{0|t}(\boldsymbol{x}_0|\boldsymbol{x}_t)} \left[\nabla_{\boldsymbol{x}_t} \log p_{t|0}(\boldsymbol{x}_t|\boldsymbol{x}_0)\right] = \mathbb{E}\left[\nabla_{\boldsymbol{x}_t} \log p_{t|0}(\boldsymbol{x}_t|\boldsymbol{x}_0)|\boldsymbol{x}_t\right]$$

- $\nabla_{\boldsymbol{x}_t} \log p_{t|0}(\boldsymbol{x}_t|\boldsymbol{x}_0)$ is explicit (forward transition): For $\boldsymbol{x}_t|\boldsymbol{x}_0 \sim \mathcal{N}(a_t\boldsymbol{x}_0, b_t^2 I_d)$,

$$\nabla_{\boldsymbol{x}_t} \log p_{t|0}(\boldsymbol{x}_t|\boldsymbol{x}_0) = \nabla_{\boldsymbol{x}_t}\left[-\frac{1}{2b_t^2}\left\|\boldsymbol{x}_t - a_t\boldsymbol{x}_0\right\|^2 + C\right] = -\frac{1}{b_t^2}(\boldsymbol{x}_t - a_t\boldsymbol{x}_0) = -\frac{1}{b_t}\boldsymbol{Z}_t$$

- But the distribution $p_{0|t}(\boldsymbol{x}_0|\boldsymbol{x}_t)$ is not explicit (backward conditional)!

$$\mathbb{E}\left[\nabla_{\boldsymbol{x}_t} \log p_{t|0}(\boldsymbol{x}_t|\boldsymbol{x}_0)|\boldsymbol{x}_t\right] = -\frac{1}{b_t^2}\left(\boldsymbol{x}_t - a_t\mathbb{E}[\boldsymbol{x}_0|\boldsymbol{x}_t]\right)$$

- $\mathbb{E}[\boldsymbol{x}_0|\boldsymbol{x}_t]$ is the best estimate of the initial noise-free $\boldsymbol{x}_0$ given its noisy version $\boldsymbol{x}_t$.

Learning the score function: Denoising score matching

$$\nabla_{\boldsymbol{x}_t} \log p_t(\boldsymbol{x}_t) = \mathbb{E}_{\boldsymbol{x}_0 \sim p_{0|t}(\boldsymbol{x}_0|\boldsymbol{x}_t)} \left[\nabla_{\boldsymbol{x}_t} \log p_{t|0}(\boldsymbol{x}_t|\boldsymbol{x}_0)\right] = \mathbb{E}\left[\nabla_{\boldsymbol{x}_t} \log p_{t|0}(\boldsymbol{x}_t|\boldsymbol{x}_0)|\boldsymbol{x}_t\right]$$

We use the following properties of the **conditional expectation**.

- $Y = \mathbb{E}[X|\mathcal{F}]$ if and only if $Y = \operatorname{argmin}\{\mathbb{E}\|X - Z\|^2,\ Z \in \mathrm{L}^2(\mathcal{F})\}$.
- $Y \in \sigma(X)$ iif there exists $\varphi:\ \mathbb{R}^d \to \mathbb{R}^d$ (measurable) with $Y = \varphi(X)$.
- $Y = \mathbb{E}[X|U]$ if $Y = \varphi(U)$ with $\varphi = \operatorname{argmin}\{\mathbb{E}\|X - \varphi(U)\|^2,\ \varphi \in \mathrm{L}^2(U)\}$.

Hence the function $\boldsymbol{x}_t \mapsto \nabla_{\boldsymbol{x}_t} \log p_t(\boldsymbol{x}_t)$ is the solution

$$\nabla_{\boldsymbol{x}_t} \log p_t = \operatorname{argmin}\{\mathbb{E}_{p_{0,t}}\|\varphi(\boldsymbol{x}_t) - \nabla_{\boldsymbol{x}_t} \log p_{t|0}(\boldsymbol{x}_t|\boldsymbol{x}_0)\|^2,\ \varphi \in \mathrm{L}^2(p_t)\}$$

- We obtain a **loss function** to learn the function φ using Monte Carlo approximation with samples $(\boldsymbol{x}_0, \boldsymbol{x}_t)$ for the expectation.

Learning the score function: Denoising score matching

$$\nabla_{\boldsymbol{x}_t} \log p_t = \operatorname{argmin}\{\mathbb{E}_{p_{0,t}} \|\varphi(\boldsymbol{x}_t) - \nabla_{\boldsymbol{x}_t} \log p_{t|0}(\boldsymbol{x}_t|\boldsymbol{x}_0)\|^2,\ \varphi \in \mathrm{L}^2(p_t)\}$$

- $\varphi : \mathbb{R}^d \to \mathbb{R}^d$ will be approximated with a neural network such as a (complex) U-net (Ho et al., 2020).
- But we need to have an approximation of $\nabla_{\boldsymbol{x}_t} \log p_t$ for all time t (at least for the times t_n in our Euler-Maruyama scheme).
- In practice we share the same network architecture for all time t: one learns a network $\boldsymbol{s}_\theta(\boldsymbol{x}, t)$ such that

$$\boldsymbol{s}_\theta(\boldsymbol{x}, t) \approx \nabla_{\boldsymbol{x}} \log p_t(\boldsymbol{x}), \quad \boldsymbol{x} \in \mathbb{R}^d,\ t \in [0, T].$$

Final loss for denoising score matching: (Song et al., 2021b)

$$\theta^* = \operatorname{argmin} \mathbb{E}_t \left(\lambda_t \mathbb{E}_{(\boldsymbol{x}_0, \boldsymbol{x}_t)} \|\boldsymbol{s}_\theta(\boldsymbol{x}_t, t) - \nabla_{\boldsymbol{x}_t} \log p_{t|0}(\boldsymbol{x}_t|\boldsymbol{x}_0)\|^2\right)$$

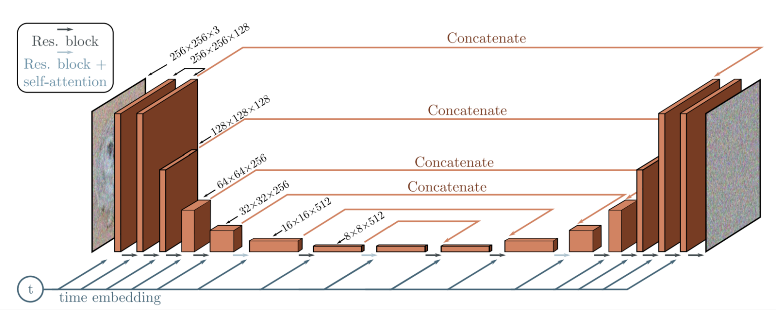
where t is chosen uniformly in $[0, T]$ and $t \mapsto \lambda_t$ is a weighting term to balance the importance of each t.

Practical aspects of diffusion models: Training and sampling

Score architecture

$$\theta^* = \operatorname{argmin} \mathbb{E}_t \left(\lambda_t \mathbb{E}_{(\boldsymbol{x}_0, \boldsymbol{x}_t)} \| s_\theta(\boldsymbol{x}_t, t) - \nabla_{\boldsymbol{x}_t} \log p_{t|0}(\boldsymbol{x}_t | \boldsymbol{x}_0) \|^2 \right)$$

- $s_\theta : \mathbb{R}^d \times [0, T] \to \mathbb{R}^d$ is a (complex) U-net (Ronneberger et al., 2015), eg in (Ho et al., 2020) “All models have two convolutional residual blocks per resolution level and self-attention blocks at the 16×16 resolution between the convolutional blocks”.
- Diffusion time t is specified by adding the Transformer sinusoidal position embedding into each residual block (Vaswani et al., 2017).

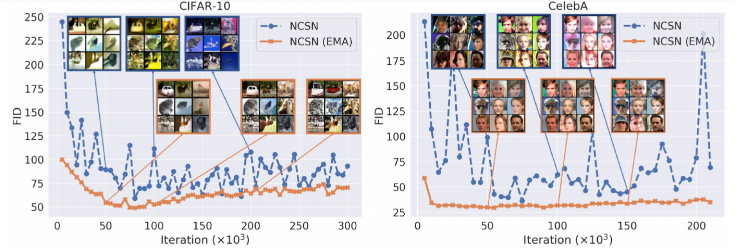


(source: learnopencv.com)

- Several choices for $t \mapsto \lambda_t$, linked to ELBO and data augmentation (Kingma and Gao, 2023).
- Training using Adam algorithm (Kingma and Ba, 2015), but still **unstable**.
- To regularize: **E**xponential **M**oving **A**verage (EMA) of weights.

$$\boxed{\bar{\theta}_{n+1} = (1-m)\bar{\theta}_n + m\theta_n.}$$

- Typically $m = 10^{-4}$ (more than 10^4 iterations are averaged).
- The final averaged parameters $\bar{\theta}_K$ are used at **sampling**.



Training instabilities (source: (Song and Ermon, 2020))

- The score function of a distribution is generally used for Langevin sampling (ULA or MALA):

$$X_{n+1} = X_n + \gamma \nabla_x \log p(X_n) + \sqrt{2\gamma} Z_n$$

- (Song et al., 2021b) propose to add one step of Langevin diffusion (same $t = t_n$) after each step Euler-Maruyama step (t_n to t_{n+1}).
- This means that we jump from one trajectory to the other, but we correct some defaults from the Euler scheme.
- This is called a Predictor-Corrector sampler.

Algorithm 2 PC sampling (VE SDE)

$\mathbf{x}_N \sim \mathcal{N}(\mathbf{0}, \sigma_{\max}^2 \mathbf{I})$
for $i = N-1$ **to** 0 **do**
  $\mathbf{x}_i' \leftarrow \mathbf{x}_{i+1} + (\sigma_{i+1}^2 - \sigma_i^2)\mathbf{s}_{\boldsymbol{\theta}^*}(\mathbf{x}_{i+1}, \sigma_{i+1})$
  $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  $\mathbf{x}_i \leftarrow \mathbf{x}_i' + \sqrt{\sigma_{i+1}^2 - \sigma_i^2}\,\mathbf{z}$
  for $j = 1$ **to** M **do**
    $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
    $\mathbf{x}_i \leftarrow \mathbf{x}_i + \epsilon_i \mathbf{s}_{\boldsymbol{\theta}^*}(\mathbf{x}_i, \sigma_i) + \sqrt{2\epsilon_i}\,\mathbf{z}$
return $\mathbf{x}_0$

Algorithm 3 PC sampling (VP SDE)

$\mathbf{x}_N \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
for $i = N-1$ **to** 0 **do**
  $\mathbf{x}_i' \leftarrow (2 - \sqrt{1-\beta_{i+1}})\mathbf{x}_{i+1} + \beta_{i+1}\mathbf{s}_{\boldsymbol{\theta}^*}(\mathbf{x}_{i+1}, i+1)$
  $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  $\mathbf{x}_i \leftarrow \mathbf{x}_i' + \sqrt{\beta_{i+1}}\,\mathbf{z}$ — Predictor
  for $j = 1$ **to** M **do** — Corrector
    $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
    $\mathbf{x}_i \leftarrow \mathbf{x}_i + \epsilon_i \mathbf{s}_{\boldsymbol{\theta}^*}(\mathbf{x}_i, i) + \sqrt{2\epsilon_i}\,\mathbf{z}$
return $\mathbf{x}_0$

(source: (Song et al., 2021b))

- (Song et al., 2021b) achieved SOTA in terms of FID for CIFAR-10 unconditional sampling.
- Very good results for 1024×1024 portrait images.
- See also "Diffusion Models Beat GANs on Image Synthesis" (Dhariwal and Nichol, 2021) (self-explanatory title).

(source: FFHQ 1024×1024 samples (Song et al., 2021b))

Many approximations in the full generative pipelines:

- The final distribution p_T is not exactly a normal distribution.
- The learnt U-net model $\boldsymbol{s}_\theta$ is far from being the exact score function: Sample-based, limitations from the architecture...
- Discrete sampling scheme (Euler-Maruyama, Predictor-Corrector,...).
- Score function may behave badly near $t = 0$ (irregular density in case of manifold hypothesis).

But we do have theoretical guarantees if all is well controled!

Theorem (Convergence guarantees (De Bortoli, 2022))
Let p_0 be the data distribution having a compact manifold support and let q_T be the generator distribution from the reversed diffusion. Under suitable hypotheses, the 1-Wasserstein distance $W_1(p_0, q_T)$ can be explicitly bounded and tends to zero when all the parameters are refined (more Euler steps, better score learning, etc.).

The deterministic approach: Probability flow ODE

Sampling via an ODE

We derived the Fokker-Planck equation for $q_t = p_{T-t}$ of reversed diffusion $\boldsymbol{y}_t = \boldsymbol{x}_{T-t}$.

$$\partial_t q_t(\boldsymbol{x}) = -\partial_t p_{T-t}(\boldsymbol{x})$$

$$= \operatorname{div}_{\boldsymbol{x}} \left(\boldsymbol{f}(\boldsymbol{x}, T-t) p_{T-t}(\boldsymbol{x})\right) - \frac{1}{2} g(T-t)^2 \Delta_{\boldsymbol{x}} p_{T-t}(\boldsymbol{x})$$

$$= \operatorname{div}_{\boldsymbol{x}} \left(\boldsymbol{f}(\boldsymbol{x}, T-t) p_{T-t}(\boldsymbol{x})\right) + \left(-1 + \frac{1}{2}\right) g(T-t)^2 \Delta_{\boldsymbol{x}} p_{T-t}(\boldsymbol{x})$$

$$= -\operatorname{div}_{\boldsymbol{x}} \left(\left[-\boldsymbol{f}(\boldsymbol{x}, T-t) + g(T-t)^2 \nabla_{\boldsymbol{x}} \log p_{T-t}(\boldsymbol{x})\right] p_{T-t}(\boldsymbol{x})\right) + \frac{1}{2} g(T-t)^2 \Delta_{\boldsymbol{x}} p_{T-t}(\boldsymbol{x})$$

This is the Fokker-Planck equation associated with the diffusion SDE:

$$d\boldsymbol{y}_t = \left[-\boldsymbol{f}(\boldsymbol{y}_t, T-t) + g(T-t)^2 \nabla_{\boldsymbol{x}} \log p_{T-t}(\boldsymbol{y}_t)\right] dt + g(T-t) d\boldsymbol{w}_t.$$

Sampling via an ODE

We derived the Fokker-Planck equation for $q_t = p_{T-t}$ of reversed diffusion $\boldsymbol{y}_t = \boldsymbol{x}_{T-t}$.

$$\begin{aligned}
&\partial_t q_t(\boldsymbol{x}) = -\partial_t p_{T-t}(\boldsymbol{x}) \\
&= \operatorname{div}_{\boldsymbol{x}}\left(\boldsymbol{f}(\boldsymbol{x}, T-t) p_{T-t}(\boldsymbol{x})\right) - \frac{1}{2} g(T-t)^2 \Delta_{\boldsymbol{x}} p_{T-t}(\boldsymbol{x}) \\
&= \operatorname{div}_{\boldsymbol{x}}\left(\boldsymbol{f}(\boldsymbol{x}, T-t) p_{T-t}(\boldsymbol{x})\right) + \left(-1 + \frac{1}{2}\right) g(T-t)^2 \Delta_{\boldsymbol{x}} p_{T-t}(\boldsymbol{x})
\end{aligned}$$

Sampling via an ODE

We derived the Fokker-Planck equation for $q_t = p_{T-t}$ of reversed diffusion $\boldsymbol{y}_t = \boldsymbol{x}_{T-t}$.

$$\begin{aligned}
&\partial_t q_t(\boldsymbol{x}) = -\partial_t p_{T-t}(\boldsymbol{x}) \\
&= \operatorname{div}_{\boldsymbol{x}}\left(\boldsymbol{f}(\boldsymbol{x}, T-t) p_{T-t}(\boldsymbol{x})\right) - \frac{1}{2} g(T-t)^2 \Delta_{\boldsymbol{x}} p_{T-t}(\boldsymbol{x}) \\
&= \operatorname{div}_{\boldsymbol{x}}\left(\boldsymbol{f}(\boldsymbol{x}, T-t) p_{T-t}(\boldsymbol{x})\right) + \left(-\frac{1}{2} + 0\right) g(T-t)^2 \Delta_{\boldsymbol{x}} p_{T-t}(\boldsymbol{x})
\end{aligned}$$

Sampling via an ODE

We derived the Fokker-Planck equation for $q_t = p_{T-t}$ of reversed diffusion $\boldsymbol{y}_t = \boldsymbol{x}_{T-t}$.

$$\begin{aligned}
\partial_t q_t(\boldsymbol{x}) &= -\partial_t p_{T-t}(\boldsymbol{x}) \\
&= \operatorname{div}_{\boldsymbol{x}} \left(\boldsymbol{f}(\boldsymbol{x}, T-t) p_{T-t}(\boldsymbol{x})\right) - \frac{1}{2} g(T-t)^2 \Delta_{\boldsymbol{x}} p_{T-t}(\boldsymbol{x}) \\
&= \operatorname{div}_{\boldsymbol{x}} \left(\boldsymbol{f}(\boldsymbol{x}, T-t) p_{T-t}(\boldsymbol{x})\right) + \left(-\frac{1}{2} + 0\right) g(T-t)^2 \Delta_{\boldsymbol{x}} p_{T-t}(\boldsymbol{x}) \\
&= -\operatorname{div}_{\boldsymbol{x}} \left(\left[-\boldsymbol{f}(\boldsymbol{x}, T-t) + \frac{1}{2} g(T-t)^2 \nabla_{\boldsymbol{x}} \log p_{T-t}(\boldsymbol{x})\right] p_{T-t}(\boldsymbol{x})\right)
\end{aligned}$$

This is the Fokker-Planck equation associated with the diffusion SDE:

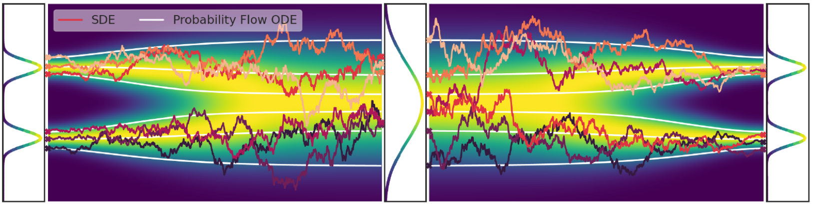
$$d\boldsymbol{y}_t = \left[-\boldsymbol{f}(\boldsymbol{y}_t, T-t) + \frac{1}{2} g(T-t)^2 \nabla_{\boldsymbol{x}} \log p_{T-t}(\boldsymbol{y}_t)\right] dt.$$

which is an **Ordinary Differential Equation (ODE)** (no stochastic term) !

Reverse diffusion via an ODE

$$dy_t = \left[-f(y_t, T-t) + \frac{1}{2} g(T-t)^2 \nabla_x \log p_{T-t}(y_t) \right] dt.$$

This ODE is called a **probability flow ODE**.



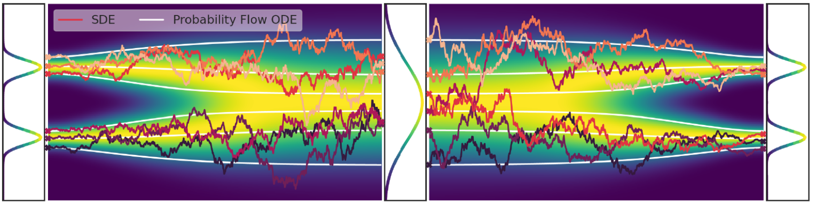
(source: (Song and Ermon, 2020))

- Like with normalizing flows, we get a deterministic mapping between initial noise and generated images.
- We do not simulate the (chaotic) path of the stochastic diffusion **but we still have the same marginal distribution** p_t.
- We can use **any ODE solver**, with higher order than Euler scheme.

Reverse diffusion via an ODE

$$dy_t = \left[-f(y_t, T-t) + \frac{1}{2} g(T-t)^2 \nabla_x \log p_{T-t}(y_t) \right] dt.$$

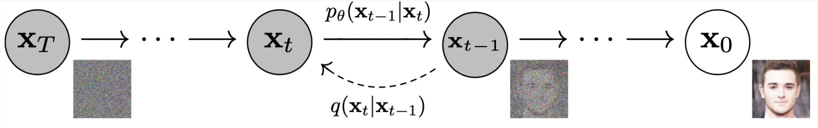
This ODE is called a **probability flow ODE**.



(source: (Song and Ermon, 2020))

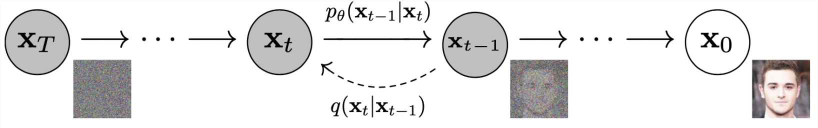
- From (Karras et al., 2022) “Through extensive tests, we have found Heun’s 2nd order method (a.k.a. improved Euler, trapezoidal rule) [...] to provide an excellent tradeoff between truncation error and NFE.”
- Requires much less NFE than stochastic samplers (eg around 50 steps instead of 1000), see also Denoising Diffusion Implicit Models (DDIM) (Song et al., 2021a) for a deterministic approach.

The discrete approach for diffusion models:
Denoising Diffusion Probabilistic Models



(source: (Ho et al., 2020))

Denoising Diffusion Probabilistic Models (**DDPM** (Ho et al., 2020)) is a discrete model with a fixed number of $T = 10^3$ steps that performs discrete diffusion.



(source: (Ho et al., 2020))

Denoising Diffusion Probabilistic Models (**DDPM** (Ho et al., 2020)) is a discrete model with a fixed number of $T = 10^3$ steps that performs discrete diffusion.

WARNING: Slight change of notation

Forward model: Discrete variance preserving diffusion

- Distribution of samples: $q(\boldsymbol{x}_0)$.
- Conditional Gaussian noise: $q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}) = \mathcal{N}(\sqrt{1-\beta_t}\boldsymbol{x}_{t-1}, \beta_t \boldsymbol{I}_d)$

$$\boldsymbol{x}_t = \sqrt{1-\beta_t}\boldsymbol{x}_{t-1} + \sqrt{\beta_t}\,\boldsymbol{z}_t$$

  where the variance schedule $(\beta_t)_{1\leq t\leq T} \in (0,1)$ is fixed.
- One step noising $q(\boldsymbol{x}_t|\boldsymbol{x}_0)$: With $\alpha_t = 1-\beta_t$ and $\bar{\alpha} = \mathrm{cumprod}(\alpha)$

$$\boldsymbol{x}_t = \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\varepsilon} \quad \text{where} \quad \boldsymbol{\varepsilon} \text{ is standard independent of } \boldsymbol{x}_0.$$

- We consider the diffusion as a fixed stochastic encoder
- We want to learn a **stochastic decoder** p_θ:

$$p_\theta(\boldsymbol{x}_{0:T}) = \underbrace{p(\boldsymbol{x}_T)}_{\text{fixed latent prior}} \prod_{t=1}^{T} \underbrace{p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)}_{\text{learnable backward transitions}} .$$

$$\text{with} \quad p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t) = \mathcal{N}(\mu_\theta(\boldsymbol{x}_t, t), \beta_t \boldsymbol{I}_d)$$

$$\text{Compare with:} \quad q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}) = \mathcal{N}(\sqrt{1-\beta_t}\boldsymbol{x}_{t-1}, \beta_t \boldsymbol{I}_d)$$

- Recall same diffusion coefficient, new backward drift to be learnt,...
- Oversimplified version compare to (Ho et al., 2020), there are ways to also learn the variance for each pixel, see (Nichol and Dhariwal, 2021).
- Then we look for training the decoder by maximizing an **ELBO**.

DDPM: Training loss

$$\mathbb{E}(-\log p_\theta(\boldsymbol{x}_0)) \leq \mathbb{E}_q \left[-\log\left[\frac{p_\theta(\boldsymbol{x}_{0:T})}{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)}\right]\right] := L$$

We have

$$L = \mathbb{E}_q \left[-\log p(\boldsymbol{x}_T) - \sum_{t=1}^{T} \log \frac{p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)}{q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})}\right]$$

$$\mathbb{E}(-\log p_\theta(\boldsymbol{x}_0)) \leq \mathbb{E}_q \left[-\log\left[\frac{p_\theta(\boldsymbol{x}_{0:T})}{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)}\right]\right] := L$$

We have

$$L = \mathbb{E}_q \left[-\log p(\boldsymbol{x}_T) - \sum_{t=1}^{T} \log \frac{p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)}{q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})}\right]$$

$$= \mathbb{E}_q\left[D_{\mathrm{KL}}(q(\boldsymbol{x}_T|\boldsymbol{x}_0)\|p(\boldsymbol{x}_T)) + \sum_{t=2}^{T} D_{\mathrm{KL}}(q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t,\boldsymbol{x}_0)\|p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)) - \log p_\theta(\boldsymbol{x}_0|\boldsymbol{x}_1)\right]$$

Computation of $D_{\mathrm{KL}}(q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t,\boldsymbol{x}_0)\|p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t))$

By Bayes rule,

$$q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t,\boldsymbol{x}_0) = q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1},\boldsymbol{x}_0)\frac{q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_0)}{q(\boldsymbol{x}_t|\boldsymbol{x}_0)} = q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})\frac{q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_0)}{q(\boldsymbol{x}_t|\boldsymbol{x}_0)}$$

Computation shows that this is a normal distribution $\mathcal{N}(\tilde{\mu}(\boldsymbol{x}_t,\boldsymbol{x}_0),\tilde{\beta}_t I_d)$ with

$$\tilde{\mu}(\boldsymbol{x}_t,\boldsymbol{x}_0) = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\boldsymbol{x}_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\boldsymbol{x}_t \quad \text{and} \quad \tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t.$$

Using the expression of the KL-divergence between Gaussian distributions,

$$D_{\mathrm{KL}}(q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t,\boldsymbol{x}_0)\|p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)) = \frac{1}{\beta_t}\|\mu_\theta(\boldsymbol{x}_t,t)-\tilde{\mu}(\boldsymbol{x}_t,\boldsymbol{x}_0)\|^2 + C$$

$$L_t = \mathbb{E}_q\left[D_{\mathrm{KL}}(q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t,\boldsymbol{x}_0)\|p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t))\right] = \frac{1}{\beta_t}\mathbb{E}_q\left[\|\mu_\theta(\boldsymbol{x}_t,t)-\tilde{\mu}(\boldsymbol{x}_t,\boldsymbol{x}_0)\|^2\right] + C$$

DDPM: Noise reparameterization

Rewrite everything **in function of the added standard noise** ε:

$$\boldsymbol{x}_t(\boldsymbol{x}_0, \boldsymbol{\varepsilon}) = \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\varepsilon}$$

Then $\mu_\theta(\boldsymbol{x}_t, t)$ must predict

$$\tilde{\mu}(\boldsymbol{x}_t, \boldsymbol{x}_0) = \frac{1}{\sqrt{\alpha_t}}\left(\boldsymbol{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\varepsilon}\right)$$

If we parameterize

$$\mu_\theta(\boldsymbol{x}_t, t) = \frac{1}{\sqrt{\alpha_t}}\left(\boldsymbol{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\varepsilon}_\theta(\boldsymbol{x}_t, t)\right)$$

Then the loss is simply

$$\begin{aligned} L_t &= \frac{\beta_t}{1-\bar{\alpha}_t}\mathbb{E}_q\left[\left\|\boldsymbol{\varepsilon}_\theta(\boldsymbol{x}_t, t) - \boldsymbol{\varepsilon}\right\|^2\right] + C \\ &= \frac{\beta_t}{1-\bar{\alpha}_t}\mathbb{E}_{\boldsymbol{x}_0,\boldsymbol{\varepsilon}}\left[\left\|\boldsymbol{\varepsilon}_\theta(\sqrt{\bar{\alpha}_t}\boldsymbol{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\varepsilon}, t) - \boldsymbol{\varepsilon}\right\|^2\right] + C \end{aligned}$$

That is we must predict the noise ε added to $\boldsymbol{x}_0$ (without knowing $\boldsymbol{x}_0$).

DDPM: Training and sampling

$$L = \mathbb{E}_q\left[D_{\mathrm{KL}}(q(\boldsymbol{x}_T|\boldsymbol{x}_0)\|p(\boldsymbol{x}_T)) + \sum_{t=2}^{T} D_{\mathrm{KL}}(q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t,\boldsymbol{x}_0)\|p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)) - \log p_\theta(\boldsymbol{x}_0|\boldsymbol{x}_1)\right]$$

$$= \sum_{t=2}^{T} L_t + L_1 + C$$

- The L_1 term is dealt differently (to account for discretization of $\boldsymbol{x}_0$).
- (Ho et al., 2020) proposes to simplify the loss (no constants):

$$L_{\text{simple}} = \mathbb{E}_{t,\boldsymbol{x}_0,\boldsymbol{\varepsilon}}\left[\left\|\boldsymbol{\varepsilon}_\theta(\sqrt{\bar{\alpha}_t}\boldsymbol{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\varepsilon}, t) - \boldsymbol{\varepsilon}\right\|^2\right]$$

Algorithm 1 Training

repeat
  $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
  $t \sim \mathrm{Uniform}(\{1,\dots,T\})$
  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0},\mathbf{I})$
  Take gradient descent step on
    $\nabla_\theta \left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t)\right\|^2$
until converged

Algorithm 2 Sampling

$\mathbf{x}_T \sim \mathcal{N}(\mathbf{0},\mathbf{I})$
for $t = T,\dots,1$ **do**
  $\mathbf{z} \sim \mathcal{N}(\mathbf{0},\mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
  $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t,t)\right) + \sigma_t\mathbf{z}$
end for
return $\mathbf{x}_0$

Recall $\sigma_t = \sqrt{\beta_t}$ here.

(source: (Ho et al., 2020))

DDPM: Denoiser

The Unet $\varepsilon_\theta(\boldsymbol{x}_t, t)$ is a (residual) denoiser that gives an estimation of the noise ε from

$$\boldsymbol{x}_t(\boldsymbol{x}_0, \boldsymbol{\varepsilon}) = \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0 + \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\varepsilon}.$$

We get the associated estimation of $\boldsymbol{x}_0$:

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

$\boldsymbol{x}_t$ $\hat{\boldsymbol{x}}_0$ $\boldsymbol{x}_0$

$t = 11$

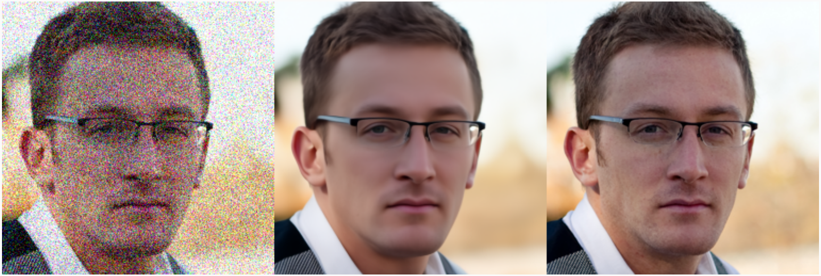
DDPM: Denoiser

The Unet $\varepsilon_\theta(\boldsymbol{x}_t, t)$ is a (residual) denoiser that gives an estimation of the noise ε from

$$\boldsymbol{x}_t(\boldsymbol{x}_0, \boldsymbol{\varepsilon}) = \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\varepsilon}.$$

We get the associated estimation of $\boldsymbol{x}_0$:

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



$\boldsymbol{x}_t$ $\qquad$ $\hat{\boldsymbol{x}}_0$ $\qquad$ $\boldsymbol{x}_0$

$t = 100$

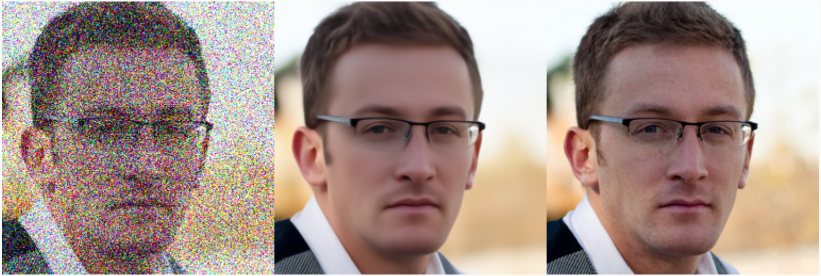
DDPM: Denoiser

The Unet $\varepsilon_\theta(\boldsymbol{x}_t, t)$ is a (residual) denoiser that gives an estimation of the noise ε from

$$\boldsymbol{x}_t(\boldsymbol{x}_0, \boldsymbol{\varepsilon}) = \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0 + \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\varepsilon}.$$

We get the associated estimation of $\boldsymbol{x}_0$:

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



$\boldsymbol{x}_t$ $\hat{\boldsymbol{x}}_0$ $\boldsymbol{x}_0$

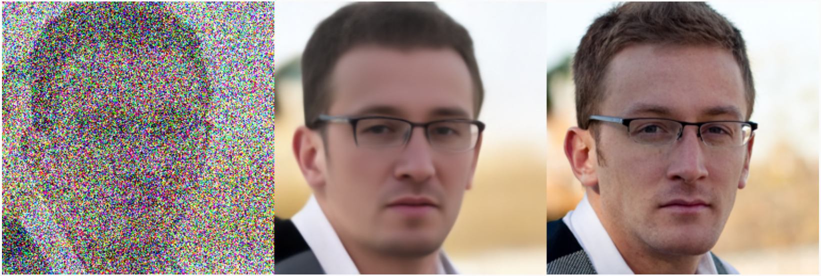
$t = 200$

The Unet $\varepsilon_\theta(\boldsymbol{x}_t, t)$ is a (residual) denoiser that gives an estimation of the noise ε from

$$\boldsymbol{x}_t(\boldsymbol{x}_0, \boldsymbol{\varepsilon}) = \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\varepsilon}.$$

We get the associated estimation of $\boldsymbol{x}_0$:

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



$\boldsymbol{x}_t$ $\hat{\boldsymbol{x}}_0$ $\boldsymbol{x}_0$

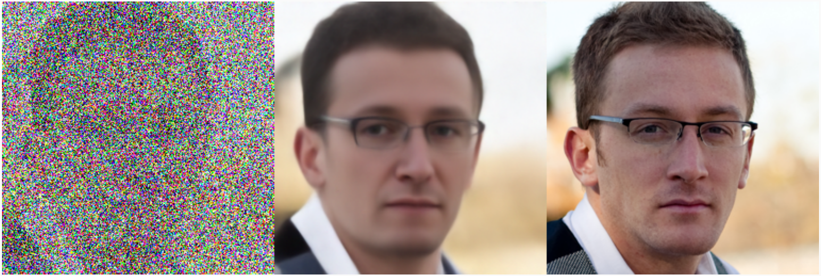
$t = 400$

The Unet $\varepsilon_\theta(\boldsymbol{x}_t, t)$ is a (residual) denoiser that gives an estimation of the noise ε from

$$\boldsymbol{x}_t(\boldsymbol{x}_0, \boldsymbol{\varepsilon}) = \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\varepsilon}.$$

We get the associated estimation of $\boldsymbol{x}_0$:

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



$\boldsymbol{x}_t$ $\hat{\boldsymbol{x}}_0$ $\boldsymbol{x}_0$

$t = 500$

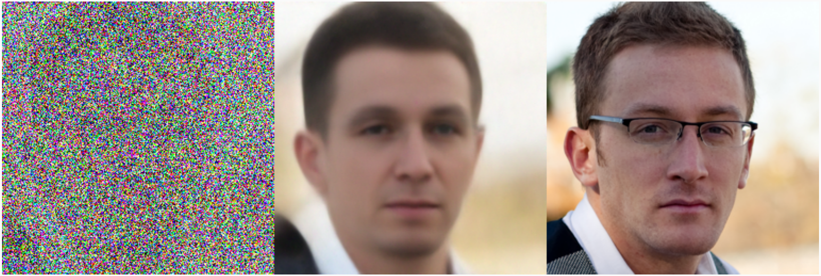
DDPM: Denoiser

The Unet $\varepsilon_\theta(\boldsymbol{x}_t, t)$ is a (residual) denoiser that gives an estimation of the noise ε from

$$\boldsymbol{x}_t(\boldsymbol{x}_0, \boldsymbol{\varepsilon}) = \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0 + \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\varepsilon}.$$

We get the associated estimation of $\boldsymbol{x}_0$:

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



$\boldsymbol{x}_t$ $\quad$ $\hat{\boldsymbol{x}}_0$ $\quad$ $\boldsymbol{x}_0$

$t = 600$

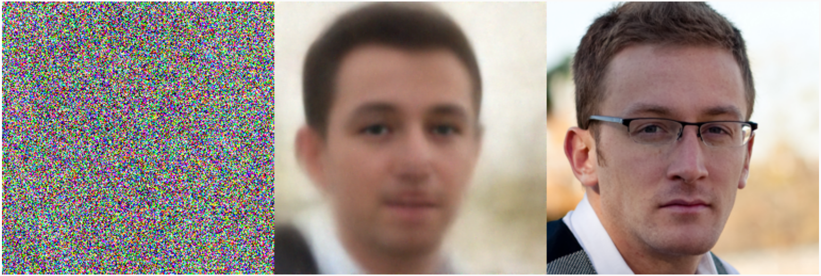
DDPM: Denoiser

The Unet $\varepsilon_\theta(\boldsymbol{x}_t, t)$ is a (residual) denoiser that gives an estimation of the noise ε from

$$\boldsymbol{x}_t(\boldsymbol{x}_0, \boldsymbol{\varepsilon}) = \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0 + \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\varepsilon}.$$

We get the associated estimation of $\boldsymbol{x}_0$:

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



$\boldsymbol{x}_t$ $\hat{\boldsymbol{x}}_0$ $\boldsymbol{x}_0$

$t = 700$

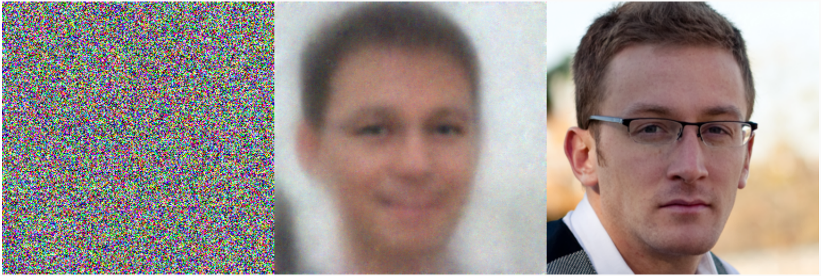
DDPM: Denoiser

The Unet $\varepsilon_\theta(\boldsymbol{x}_t, t)$ is a (residual) denoiser that gives an estimation of the noise ε from

$$\boldsymbol{x}_t(\boldsymbol{x}_0, \boldsymbol{\varepsilon}) = \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0 + \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\varepsilon}.$$

We get the associated estimation of $\boldsymbol{x}_0$:

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



$\boldsymbol{x}_t$ $\qquad$ $\hat{\boldsymbol{x}}_0$ $\qquad$ $\boldsymbol{x}_0$

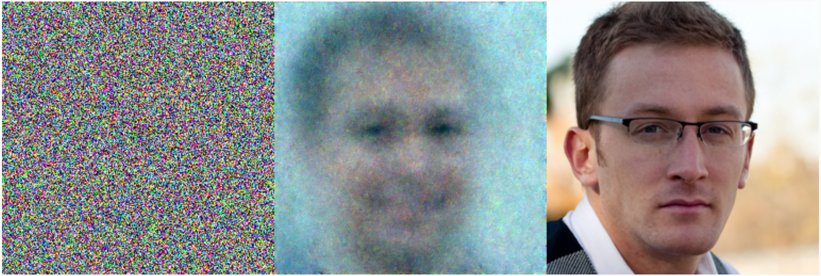
$t = 800$

The Unet $\varepsilon_\theta(\boldsymbol{x}_t, t)$ is a (residual) denoiser that gives an estimation of the noise ε from

$$\boldsymbol{x}_t(\boldsymbol{x}_0, \boldsymbol{\varepsilon}) = \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0 + \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\varepsilon}.$$

We get the associated estimation of $\boldsymbol{x}_0$:

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



$\boldsymbol{x}_t$ $\hat{\boldsymbol{x}}_0$ $\boldsymbol{x}_0$

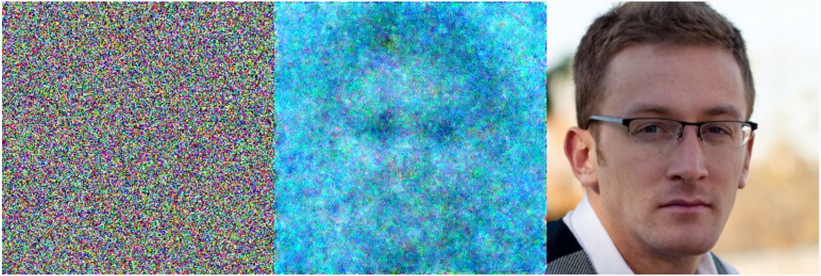
$t = 900$

The Unet $\varepsilon_\theta(\boldsymbol{x}_t, t)$ is a (residual) denoiser that gives an estimation of the noise ε from

$$\boldsymbol{x}_t(\boldsymbol{x}_0, \boldsymbol{\varepsilon}) = \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0 + \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\varepsilon}.$$

We get the associated estimation of $\boldsymbol{x}_0$:

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



$\boldsymbol{x}_t$ $\hat{\boldsymbol{x}}_0$ $\boldsymbol{x}_0$

$t = 1000$

DDPM: Sampling

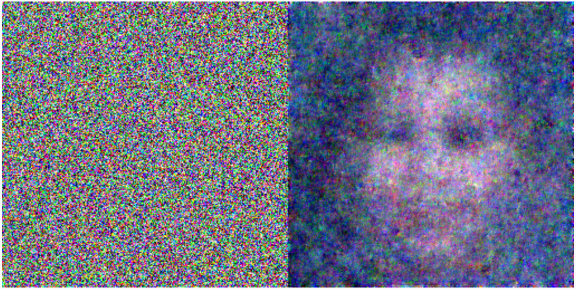
Algorithm 1 Training

repeat
  $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
  $t \sim \mathrm{Uniform}(\{1, \dots, T\})$
  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  Take gradient descent step on
  $$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
until converged

Algorithm 2 Sampling

$\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
for $t = T, \dots, 1$ **do**
  $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
  $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right) + \sigma_t \mathbf{z}$
end for
return $\mathbf{x}_0$

Recall $\sigma_t = \sqrt{\beta_t}$ here. (source: (Ho et al., 2020))



$\boldsymbol{x}_t$ $\hat{\boldsymbol{x}}_0$

$t = 999$

Algorithm 1 Training

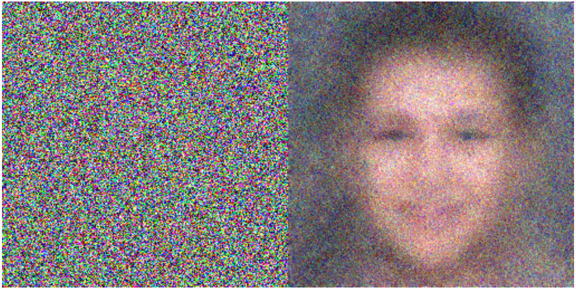
1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \mathrm{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

Recall $\sigma_t = \sqrt{\beta_t}$ here.

(source: (Ho et al., 2020))



DDPM: Sampling

Algorithm 1 Training

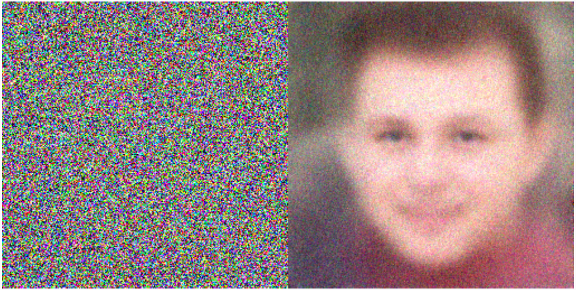
repeat
  $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
  $t \sim \mathrm{Uniform}(\{1, \dots, T\})$
  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  Take gradient descent step on
  $$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
until converged

Algorithm 2 Sampling

$\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
for $t = T, \dots, 1$ **do**
  $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
  $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right) + \sigma_t \mathbf{z}$
end for
return $\mathbf{x}_0$

Recall $\sigma_t = \sqrt{\beta_t}$ here.

(source: (Ho et al., 2020))



$\boldsymbol{x}_t$ $\hat{\boldsymbol{x}}_0$

$t = 800$

DDPM: Sampling

Algorithm 1 Training

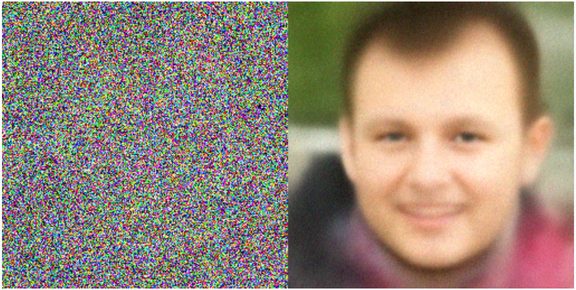
1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \mathrm{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

Recall $\sigma_t = \sqrt{\beta_t}$ here.

(source: (Ho et al., 2020))



$\boldsymbol{x}_t$ $\quad$ $\hat{\boldsymbol{x}}_0$

$t = 700$

Algorithm 1 Training

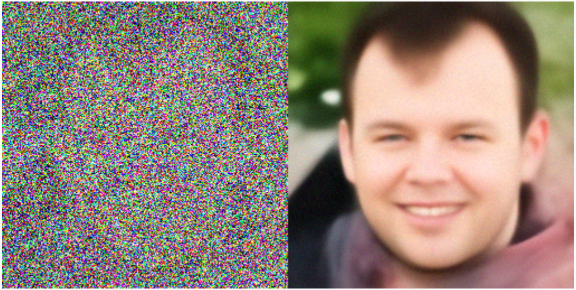
repeat
  $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
  $t \sim \mathrm{Uniform}(\{1, \dots, T\})$
  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  Take gradient descent step on
    $\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$
until converged

Algorithm 2 Sampling

$\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
for $t = T, \dots, 1$ **do**
  $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
  $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
end for
return $\mathbf{x}_0$

Recall $\sigma_t = \sqrt{\beta_t}$ here.

(source: (Ho et al., 2020))



$\boldsymbol{x}_t$ $\hat{\boldsymbol{x}}_0$

$t = 600$

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \mathrm{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

Recall $\sigma_t = \sqrt{\beta_t}$ here.

(source: (Ho et al., 2020))

x_t $\hat{x}_0$

$t = 500$

Algorithm 1 Training

repeat
  $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
  $t \sim \mathrm{Uniform}(\{1, \dots, T\})$
  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  Take gradient descent step on
  $$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
until converged

Algorithm 2 Sampling

$\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
for $t = T, \dots, 1$ **do**
  $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
  $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right) + \sigma_t \mathbf{z}$
end for
return $\mathbf{x}_0$

Recall $\sigma_t = \sqrt{\beta_t}$ here.

(source: (Ho et al., 2020))

$\boldsymbol{x}_t$ $\hat{\boldsymbol{x}}_0$

$t = 400$

DDPM: Sampling

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \mathrm{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

Recall $\sigma_t = \sqrt{\beta_t}$ here. (source: (Ho et al., 2020))

$\boldsymbol{x}_t$ $\hat{\boldsymbol{x}}_0$

$t = 300$

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \mathrm{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

Recall $\sigma_t = \sqrt{\beta_t}$ here. (source: (Ho et al., 2020))

$\boldsymbol{x}_t$ $\hat{\boldsymbol{x}}_0$

$t = 200$

Algorithm 1 Training

repeat
  $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
  $t \sim \mathrm{Uniform}(\{1, \dots, T\})$
  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  Take gradient descent step on
    $\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$
until converged

Algorithm 2 Sampling

$\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
for $t = T, \dots, 1$ **do**
  $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
  $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
end for
return $\mathbf{x}_0$

Recall $\sigma_t = \sqrt{\beta_t}$ here.

(source: (Ho et al., 2020))

$\boldsymbol{x}_t$ $\hat{\boldsymbol{x}}_0$

$t = 100$

DDPM: Sampling

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \mathrm{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

Recall $\sigma_t = \sqrt{\beta_t}$ here.

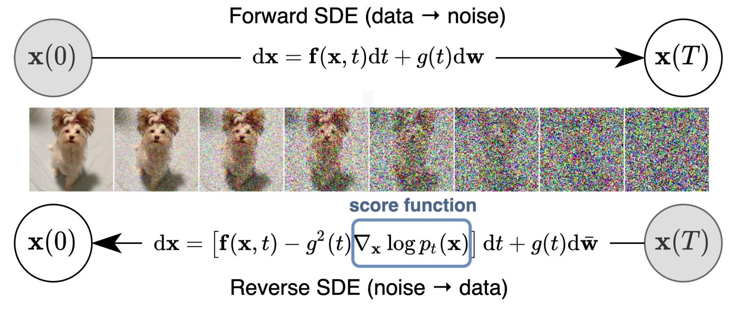
(source: (Ho et al., 2020))

$\boldsymbol{x}_t$ $\hat{\boldsymbol{x}}_0$

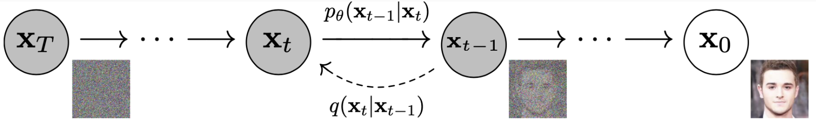
$t = 0$

Continuous and discrete diffusion models

Diffusion model via SDE: (Song et al., 2021b)



Diffusion model via Denoising Diffusion Probabilistic Models (DDPM):
(Ho et al., 2020) Discrete model with a fixed number of $T = 10^3$.



Continuous diffusion models

Forward diffusion:

$$d\boldsymbol{x}_t = \boldsymbol{f}(\boldsymbol{x}_t, t)dt + g(t)d\boldsymbol{w}_t$$

Backward diffusion: $\boldsymbol{y}_t = \boldsymbol{x}_{T-t}$

$$d\boldsymbol{y}_t = \left[-\boldsymbol{f}(\boldsymbol{y}_t, T-t) + g(T-t)^2 \nabla_{\boldsymbol{x}} \log p_{T-t}(\boldsymbol{y}_t)\right] dt + g(T-t)d\boldsymbol{w}_t.$$

- Learn score by denoising score matching:

$$\boxed{\theta^\star = \operatorname{argmin} \mathbb{E}_t \left(\lambda_t \mathbb{E}_{(\boldsymbol{x}_0, \boldsymbol{x}_t)} \|s_\theta(\boldsymbol{x}_t, t) - \nabla_{\boldsymbol{x}_t} \log p_{t|0}(\boldsymbol{x}_t | \boldsymbol{x}_0)\|^2\right)} \text{ with } t \sim \text{Unif}([0, T])$$

- Generate samples by SDE discrete scheme (e.g. Euler-Maruyama):

$$\boldsymbol{Y}_{n-1} = \boldsymbol{Y}_n - h\boldsymbol{f}(\boldsymbol{Y}_n, t_n) + hg(t_n)^2 \boldsymbol{s}_\theta(\boldsymbol{Y}_n, t_n) + g(t_n)\sqrt{h}\boldsymbol{Z}_n \quad \text{with} \quad \boldsymbol{Z}_n \sim \mathcal{N}(\boldsymbol{0}, I_d)$$

- Associated deterministic probability flow:

$$d\boldsymbol{y}_t = \left[-\boldsymbol{f}(\boldsymbol{y}_t, T-t) + \frac{1}{2} g(T-t)^2 \nabla_{\boldsymbol{x}} \log p_{T-t}(\boldsymbol{y}_t)\right] dt$$

Denoising Diffusion Probabilistic Models (DDPM)

Forward diffusion:

$$q(x_{0:T}) = \underbrace{q(\boldsymbol{x}_0)}_{\text{data distribution}} \prod_{t=1}^{T} \underbrace{q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})}_{\text{fixed forward transitions}} \quad \text{with} \quad q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}) = \mathcal{N}(\sqrt{1-\beta_t}\boldsymbol{x}_{t-1}, \beta_t I_d)$$

Backward diffusion: **stochastic decoder** p_θ:

$$p_\theta(\boldsymbol{x}_{0:T}) = \underbrace{p(\boldsymbol{x}_T)}_{\text{fixed latent prior}} \prod_{t=1}^{T} \underbrace{p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)}_{\text{learnt backward transitions}} \quad \text{with} \quad \underbrace{p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t) = \mathcal{N}(\mu_\theta(\boldsymbol{x}_t, t), \beta_t I_d)}_{\text{Gaussian approximation of } q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)}$$

- Learn the score by minimizing the ELBO (like for VAE): This boils down to denoising the diffusion iterations $\boldsymbol{x}_t = \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\varepsilon}$:

$$\theta^\star = \operatorname{argmin} \sum_{t=1}^{T} \frac{\beta_t}{1-\bar{\alpha}_t} \mathbb{E}_q \left[\|\boldsymbol{\varepsilon}_\theta(\boldsymbol{x}_t, t) - \boldsymbol{\varepsilon}\|^2 \right] + C$$

- Sampling through the stochastic decoder with

$$\mu_\theta(\boldsymbol{x}_t, t) = \frac{1}{\sqrt{\alpha_t}} \left(\boldsymbol{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\varepsilon}_\theta(\boldsymbol{x}_t, t) \right)$$

DDPM training and score matching

Posterior mean training: Recall that $\mu_\theta(\boldsymbol{x}_t, t)$ minimizes

$$\mathbb{E}_q\left[D_{\mathrm{KL}}(q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{x}_0) \| p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t))\right] = \frac{1}{\beta_t}\mathbb{E}_q\left[\|\mu_\theta(\boldsymbol{x}_t, t) - \tilde{\mu}(\boldsymbol{x}_t, \boldsymbol{x}_0)\|^2\right] + C$$

where $\tilde{\mu}(\boldsymbol{x}_t, \boldsymbol{x}_0)$ is the mean of $q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{x}_0)$. Hence ideally,

$$\mu_\theta(\boldsymbol{x}_t, t) = \mathbb{E}\left[\tilde{\mu}(\boldsymbol{x}_t, \boldsymbol{x}_0)|\boldsymbol{x}_t\right] = \mathbb{E}\left[\mathbb{E}\left[\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{x}_0\right]|\boldsymbol{x}_t\right] = \mathbb{E}\left[\boldsymbol{x}_{t-1}|\boldsymbol{x}_t\right].$$

Noise prediction training: $\boldsymbol{\varepsilon}_\theta(\boldsymbol{x}_t, t)$ minimizes

$$\mathbb{E}_q\left[\|\boldsymbol{\varepsilon}_\theta(\boldsymbol{x}_t, t) - \boldsymbol{\varepsilon}\|^2\right]$$

where $\boldsymbol{\varepsilon}$ is a function of $(\boldsymbol{x}_t, \boldsymbol{x}_0)$ (since $\boldsymbol{x}_t = \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\varepsilon}$). Hence ideally,

$$\boldsymbol{\varepsilon}_\theta(\boldsymbol{x}_t, t) = \mathbb{E}\left[\boldsymbol{\varepsilon}|\boldsymbol{x}_t\right]$$

Score matching training: Ideally,

$$s_\theta(\boldsymbol{x}_t, t) = \nabla_{\boldsymbol{x}_t}\log p_t(\boldsymbol{x}_t) = \mathbb{E}\left[\nabla_{\boldsymbol{x}_t}\log p_{t|0}(\boldsymbol{x}_t|\boldsymbol{x}_0)|\boldsymbol{x}_t\right]$$

We derived the formulas for DDPM training without considering the score function... but denoising and score functions are linked by **Tweedie formulas**:

Theorem (Tweedie formulas)
If $\boldsymbol{Y} = a\boldsymbol{X} + \sigma\boldsymbol{Z}$ with $\boldsymbol{Z} \sim \mathcal{N}(\mathbf{0}, I_d)$ independent of $\boldsymbol{X}$, $a > 0$, $\sigma > 0$, then

Tweedie denoiser:

$$\boxed{\mathbb{E}[\boldsymbol{X}|\boldsymbol{Y}] = \frac{1}{a}\left(\boldsymbol{Y} + \sigma^2 \nabla_{\boldsymbol{y}} \log p_{\boldsymbol{Y}}(\boldsymbol{Y})\right)}$$

Tweedie noise predictor:

$$\boxed{\mathbb{E}[\boldsymbol{Z}|\boldsymbol{Y}] = -\sigma \nabla_{\boldsymbol{y}} \log p_{\boldsymbol{Y}}(\boldsymbol{Y})}$$

DDPM and Tweedie

If $Y = aX + \sigma Z$, **Tweedie denoiser:** $\mathbb{E}[X|Y] = \frac{1}{a}\left(Y + \sigma^2 \nabla_y \log p_Y(Y)\right)$

Tweedie noise predictor: $\mathbb{E}[Z|Y] = -\sigma \nabla_y \log p_Y(Y)$

Tweedie for noise prediction: Predict the noise ε from x_t:

$$x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t}\varepsilon \quad \Rightarrow \quad \boxed{\mathbb{E}\left[\varepsilon | x_t\right] = -\sqrt{1 - \bar{\alpha}_t} \nabla_{x_t} \log p_t(x_t)}$$

Tweedie for one-step denoising: Predict x_{t-1} from x_t:

$$x_t = \sqrt{\alpha_t}\, x_{t-1} + \sqrt{\beta_t}\, z_t \quad \Rightarrow \quad \mathbb{E}[x_{t-1}|x_t] = \frac{1}{\sqrt{\alpha_t}}\left(x_t + \beta_t \nabla_{x_t} \log p_t(x_t)\right)$$

$$\boxed{\mathbb{E}[x_{t-1}|x_t] = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}} \mathbb{E}\left[\varepsilon | x_t\right]\right)}$$

If $Y = aX + \sigma Z$, **Tweedie denoiser:** $$\mathbb{E}[X|Y] = \frac{1}{a}\left(Y + \sigma^2 \nabla_y \log p_Y(Y)\right)$$

Tweedie noise predictor: $$\mathbb{E}[Z|Y] = -\sigma \nabla_y \log p_Y(Y)$$

Tweedie for noise prediction: Predict the noise ε from x_t:

$$x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t}\varepsilon \quad \Rightarrow \quad \mathbb{E}\left[\varepsilon | x_t\right] = -\sqrt{1 - \bar{\alpha}_t} \nabla_{x_t} \log p_t(x_t)$$

Tweedie for one-step denoising: Predict x_{t-1} from x_t:

$$x_t = \sqrt{\alpha_t}\, x_{t-1} + \sqrt{\beta_t}\, z_t \quad \Rightarrow \quad \mathbb{E}[x_{t-1}|x_t] = \frac{1}{\sqrt{\alpha_t}}\left(x_t + \beta_t \nabla_{x_t} \log p_t(x_t)\right)$$

$$\mathbb{E}[x_{t-1}|x_t] = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}} \mathbb{E}\left[\varepsilon | x_t\right]\right)$$

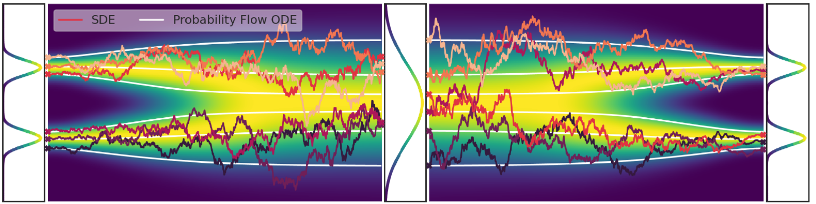
$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}} \varepsilon_\theta(x_t, t)\right)$$

Remarks: We recover the expression of $\mu_\theta(x_t, t)$ without using the one of

$$\tilde{\mu}(x_t, x_0) = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}} \varepsilon\right)$$

To sum up:

- The three trainings strategies are the same (up to weighting constants).
- The only difference between the continuous SDE model and the discrete DDPM model are the time values: $t \in [0, T]$ VS. $t = 1, \ldots, T = 10^3$.
- **Good news:** We can train a DDPM and use it for a deterministic probability flow ODE (see also the DDIM model (Song et al., 2021a) for stochastic to deterministic in the discrete case).



(source: (Song and Ermon, 2020))

Diffusion models for imaging inverse problems

Diffusion posterior sampling

We present **Diffusion Posterior Sampling (DPS)** for general noisy inverse problems (Chung et al., 2023)

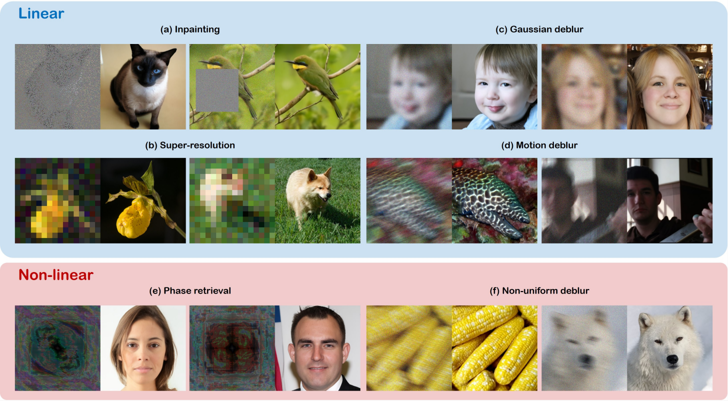


Figure 1: Solving noisy linear, and nonlinear inverse problems with diffusion models. Our reconstruction results (right) from the measurements (left) are shown.

(source: (Chung et al., 2023))

See also (Song et al., 2023), (Kawar et al., 2022) for alternative methods.

Conditional sampling

Let A be a linear operator from an inverse problem (masking operator for inpainting, blur operator for deblurring, subsampling for SR, ...).

Given some observation

$$\boldsymbol{y} = A\boldsymbol{x}_{\text{unknown}} + \boldsymbol{n}$$

where $\boldsymbol{n}$ is some additive white Gaussian noise with variance σ^2, we would like to sample

$$p_0(\boldsymbol{x}_0 | A\boldsymbol{x}_0 + \boldsymbol{n} = \boldsymbol{y}) = p_0(\boldsymbol{x}_0 | \boldsymbol{y})$$

to estimate $\boldsymbol{x}_{\text{unknown}}$ in accordance with the prior of the generative model.

Conditional sampling

From (Song et al., 2021b), we can consider the SDE for the conditional distribution $p_0(\boldsymbol{x}_0|\boldsymbol{y})$:

Backward diffusion for VP-SDE: $\boldsymbol{y}_t = \boldsymbol{x}_{T-t}$

$$d\boldsymbol{y}_t = [\beta_{T-t}\boldsymbol{y}_t + \beta_{T-t}\nabla_{\boldsymbol{x}=\boldsymbol{y}_t} \log p_{T-t}(\boldsymbol{y}_t)]\, dt + \beta_{T-t} d\boldsymbol{w}_t.$$

Conditional backward diffusion for VP-SDE: $\boldsymbol{y}_t = \boldsymbol{x}_{T-t}$

$$d\boldsymbol{y}_t = [\beta_{T-t}\boldsymbol{y}_t + \beta_{T-t}\nabla_{\boldsymbol{x}=\boldsymbol{y}_t} \log p_{T-t}(\boldsymbol{y}_t|\boldsymbol{y})]\, dt + \beta_{T-t} d\boldsymbol{w}_t.$$

By Bayes rule:

$$\log p_{T-t}(\boldsymbol{y}_t|\boldsymbol{y}) = \log p_{T-t}(\boldsymbol{y}|\boldsymbol{y}_t) + \log(p_{T-t}(\boldsymbol{y}_t)) - \log(p_{T-t}(\boldsymbol{y}))$$

Thus,

$$\nabla_{\boldsymbol{x}=\boldsymbol{y}_t} \log p_{T-t}(\boldsymbol{y}_t|\boldsymbol{y}) = \underbrace{\nabla_{\boldsymbol{x}=\boldsymbol{y}_t} \log p_{T-t}(\boldsymbol{y}|\boldsymbol{y}_t)}_{\text{intractable}} + \underbrace{\nabla_{\boldsymbol{x}=\boldsymbol{y}_t} \log(p_{T-t}(\boldsymbol{y}_t))}_{\text{usual score function}}$$

For clarity, let us write the new term with forward notation:

$$\nabla_{\boldsymbol{x}=\boldsymbol{y}_t} \log p_{T-t}(\boldsymbol{y}|\boldsymbol{y}_t) = \nabla_{\boldsymbol{x}=\boldsymbol{x}_t} \log p_t(\boldsymbol{y}|\boldsymbol{x}_t)$$

Conditional sampling

(Chung et al., 2023) propose the following approximation:

$$\log p_t(\boldsymbol{y}|\boldsymbol{x}_t) \approx \log p_t(\boldsymbol{y}|\boldsymbol{x}_0 = \hat{\boldsymbol{x}}_0(\boldsymbol{x}_t, t))$$

with $\hat{\boldsymbol{x}}_0(\boldsymbol{x}_t, t)$ the estimate of the original image from the network.

Since

$$p(\boldsymbol{y}|\boldsymbol{x}_0) = \frac{1}{(2\pi\sigma^2)^{\frac{n}{2}}} \exp\left(-\frac{\|\boldsymbol{y} - A\boldsymbol{x}_0\|^2}{2\sigma^2}\right)$$

we finally approximate

$$\nabla_{\boldsymbol{x}=\boldsymbol{x}_t} \log p_t(\boldsymbol{y}|\boldsymbol{x}_t) = -\frac{1}{2\sigma^2} \nabla_{\boldsymbol{x}_t} \|\boldsymbol{y} - A\hat{\boldsymbol{x}}_0(\boldsymbol{x}_t, t)\|^2$$

- Computing $\nabla_{\boldsymbol{x}_t} \|\boldsymbol{y} - A\hat{\boldsymbol{x}}_0(\boldsymbol{x}_t, t)\boldsymbol{x}_0\|^2$ involves a backpropagation through the Unet.
- One can expect this approximate conditional sampling to be twice as long as the sampling procedure.

Diffusion posterior sampling

Algorithm 1 DPS - Gaussian

Require: N, $\boldsymbol{y}$, $\{\zeta_i\}_{i=1}^N$, $\{\tilde{\sigma}_i\}_{i=1}^N$

1: $\boldsymbol{x}_N \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$
2: **for** $i = N-1$ **to** 0 **do**
3: $\quad \hat{\boldsymbol{s}} \leftarrow \boldsymbol{s}_\theta(\boldsymbol{x}_i, i)$
4: $\quad \hat{\boldsymbol{x}}_0 \leftarrow \frac{1}{\sqrt{\bar{\alpha}_i}}(\boldsymbol{x}_i + (1-\bar{\alpha}_i)\hat{\boldsymbol{s}})$
5: $\quad \boldsymbol{z} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$
6: $\quad \boldsymbol{x}'_{i-1} \leftarrow \frac{\sqrt{\alpha_i}(1-\bar{\alpha}_{i-1})}{1-\bar{\alpha}_i}\boldsymbol{x}_i + \frac{\sqrt{\bar{\alpha}_{i-1}}\beta_i}{1-\bar{\alpha}_i}\hat{\boldsymbol{x}}_0 + \tilde{\sigma}_i \boldsymbol{z}$
7: $\quad \boldsymbol{x}_{i-1} \leftarrow \boldsymbol{x}'_{i-1} - \zeta_i \nabla_{\boldsymbol{x}_i} \|\boldsymbol{y} - \mathcal{A}(\hat{\boldsymbol{x}}_0)\|_2^2$
8: **end for**
9: **return** $\hat{\mathbf{x}}_0$

(source: (Chung et al., 2023))

- Usual DDPM sampling (notation with $\hat{\boldsymbol{x}}_0(\boldsymbol{x}_t, t)$ instead of $\varepsilon_\theta(\boldsymbol{x}_t, t)$.

$$\mu_\theta(\boldsymbol{x}_t, t) = \frac{1}{\sqrt{\alpha_t}}\left(\boldsymbol{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\varepsilon_\theta(\boldsymbol{x}_t, t)\right) = \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\boldsymbol{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\hat{\boldsymbol{x}}_0(\boldsymbol{x}_t, t)$$

- Add a correction term to drive $A\hat{\boldsymbol{x}}_0(\boldsymbol{x}_t, t)$ close to $\boldsymbol{y}$.
- In practice $\zeta_i = \zeta_t \propto \|\boldsymbol{y} - A\hat{\boldsymbol{x}}_0(\boldsymbol{x}_t, t)\|^{-1}$.

- Very good results in terms of perceptual metric (LPIPS).

Inpainting:

$\boldsymbol{x}_{\text{unknown}}$ $\boldsymbol{y}$ $\hat{\boldsymbol{x}}_0$ $\boldsymbol{x}_t$

$t = 999$

- Very good results in terms of perceptual metric (LPIPS).

Inpainting:

x_{unknown} y $\hat{x}_0$ x_t

$t = 900$

- Very good results in terms of perceptual metric (LPIPS).

Inpainting:

$\boldsymbol{x}_{\text{unknown}}$ $\boldsymbol{y}$ $\hat{\boldsymbol{x}}_0$ $\boldsymbol{x}_t$

$t = 800$

- Very good results in terms of perceptual metric (LPIPS).

Inpainting:

x_{unknown} y $\hat{x}_0$ x_t

$t = 700$

- Very good results in terms of perceptual metric (LPIPS).

Inpainting:

x_{unknown} y $\hat{x}_0$ x_t

$t = 600$

- Very good results in terms of perceptual metric (LPIPS).

Inpainting:

$\boldsymbol{x}_{\text{unknown}}$ $\boldsymbol{y}$ $\hat{\boldsymbol{x}}_0$ $\boldsymbol{x}_t$

$t = 500$

- Very good results in terms of perceptual metric (LPIPS).

Inpainting:

x_{unknown} y $\hat{x}_0$ x_t

$t = 400$

- Very good results in terms of perceptual metric (LPIPS).

Inpainting:

$\boldsymbol{x}_{\text{unknown}}$ $\boldsymbol{y}$ $\hat{\boldsymbol{x}}_0$ $\boldsymbol{x}_t$

$t = 300$

- Very good results in terms of perceptual metric (LPIPS).

Inpainting:

$\boldsymbol{x}_{\text{unknown}}$ $\boldsymbol{y}$ $\hat{\boldsymbol{x}}_0$ $\boldsymbol{x}_t$

$t = 200$

- Very good results in terms of perceptual metric (LPIPS).

Inpainting:

$\boldsymbol{x}_{\text{unknown}}$ $\boldsymbol{y}$ $\hat{\boldsymbol{x}}_0$ $\boldsymbol{x}_t$

$t = 100$

- Very good results in terms of perceptual metric (LPIPS).

Inpainting:

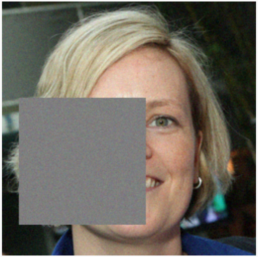
$\boldsymbol{x}_{\text{unknown}}$ $\boldsymbol{y}$ $\hat{\boldsymbol{x}}_0$ $\boldsymbol{x}_t$

$t = 0$

Diffusion posterior sampling: Inpainting results

- Very good results in terms of perceptual metric (LPIPS).
- Lack of symmetry.
- It can sometimes be really bad though!

original $\boldsymbol{x}_{\text{unknown}}$



input $\boldsymbol{y}$

output $\boldsymbol{x}_0$

- Very good results in terms of perceptual metric (LPIPS).
- Lack of symmetry.
- It can sometimes be really bad though!

original $\boldsymbol{x}_{\text{unknown}}$

input $\boldsymbol{y}$

output $\boldsymbol{x}_0$

Diffusion posterior sampling: Inpainting results

- Very good results in terms of perceptual metric (LPIPS).
- Lack of symmetry.
- It can sometimes be really bad though!

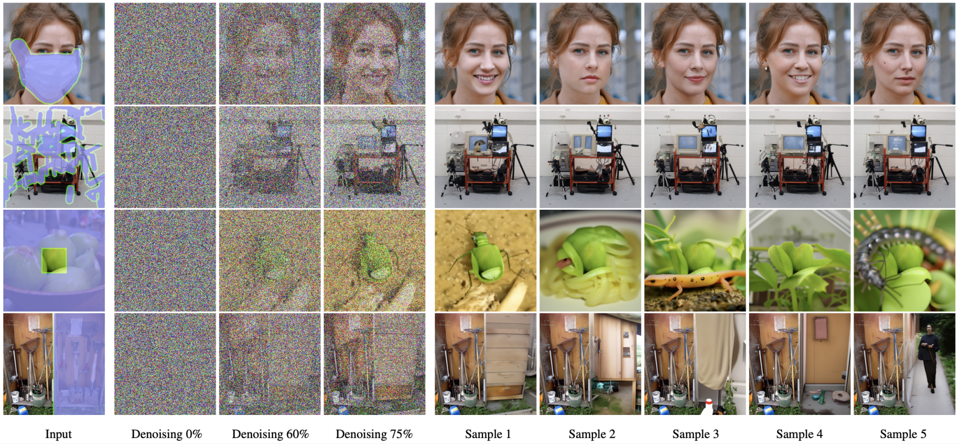
original $\boldsymbol{x}_{\text{unknown}}$

input $\boldsymbol{y}$

output $\boldsymbol{x}_0$

Diffusion posterior sampling: Inpainting results

- For inpainting it can help to go back and forth in the diffusion process (Lugmayr et al., 2022).



(source: (Lugmayr et al., 2022))

- Super-resolution with a factor $\times 4$.
- Very good results in terms of perceptual metric (LPIPS).
- Loss of details (skin defaults, etc.).

original $\boldsymbol{x}_{\text{unknown}}$

input $\boldsymbol{y}$

output $\boldsymbol{x}_0$

- Super-resolution with a factor $\times 4$.
- Very good results in terms of perceptual metric (LPIPS).
- Loss of details (skin defaults, etc.).

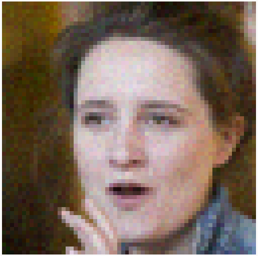
original $\boldsymbol{x}_{\text{unknown}}$

input $\boldsymbol{y}$

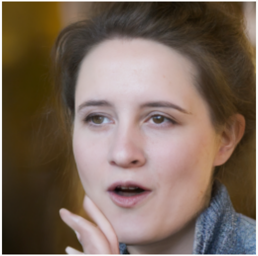
output $\boldsymbol{x}_0$

- Super-resolution with a factor $\times 4$.
- Very good results in terms of perceptual metric (LPIPS).
- Loss of details (skin defaults, etc.).

original $\boldsymbol{x}_{\text{unknown}}$



input $\boldsymbol{y}$



output $\boldsymbol{x}_0$

Conditional DDPM for super-resolution

- Super-resolution is often used to improve the quality of generated images.
- One can train a specific DDPM for this task by conditioning the Unet with the low resolution image $\varepsilon_\theta(\boldsymbol{x}_t, \boldsymbol{y}_{\mathrm{LR}}, t)$.

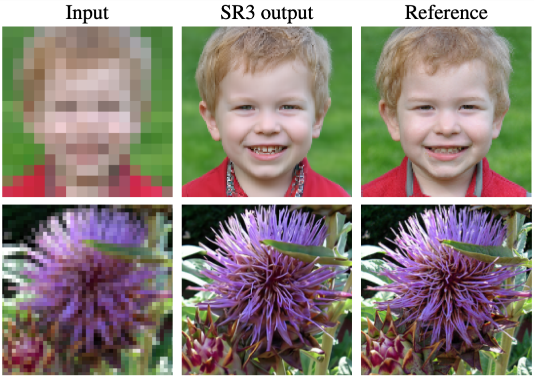
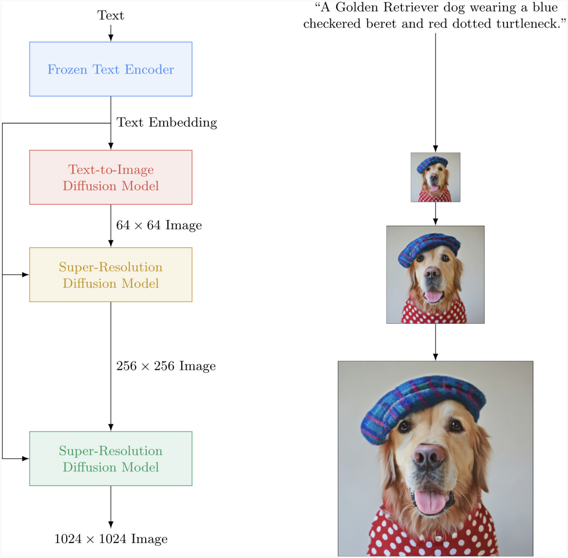


Figure 1: Two representative SR3 outputs: (top) 8× face super-resolution at 16×16→128×128 pixels (bottom) 4× natural image super-resolution at 64×64→256×256 pixels.

From (Saharia et al., 2023):
"To condition the model on the input $\boldsymbol{y}_{\mathrm{LR}}$, we upsample the low-resolution image to the target resolution using bicubic interpolation. The result is concatenated with $\boldsymbol{x}_t$ along the channel dimension."

Conditional DDPM for super-resolution

Imagen pipeline:
Text conditioning
&
Conditional
super-resolution
via DDPM



(source: (Saharia et al., 2022))

Latent diffusion model

Latent diffusion model

The pace of AI...

- **June 2022:** Publication of the **stable diffusion** paper (Rombach et al., 2022) that introduces latent diffusion models.
- **Aug. 2022:** First public release of **stable diffusion** by stability AI: Open-source text-to-image model (contrary to concurrent closed source methods DALL·E 2 by OpenAI and Imagen by Google).
- **Stable diffusion becomes a foundation model:** Used by a lot of researchers in computer vision:
  - Faster sampling/distillation
  - Better control
  - Insert specific objects
  - New areas: Videos, 3D objects, molecule generation, ...
- **Dec. 2023:** Tutorial on **Latent diffusion model** at NeurIPS 2023: Full slides here: `https://neurips2023-ldm-tutorial.github.io/`

NeurIPS 2023 Tutorial

Latent Diffusion Models:

Is the Generative AI Revolution Happening in Latent Space?

Karsten Kreis

Ruiqi Gao

Arash Vahdat

NVIDIA

Google DeepMind

https://neurips2023-ldm-tutorial.github.io/

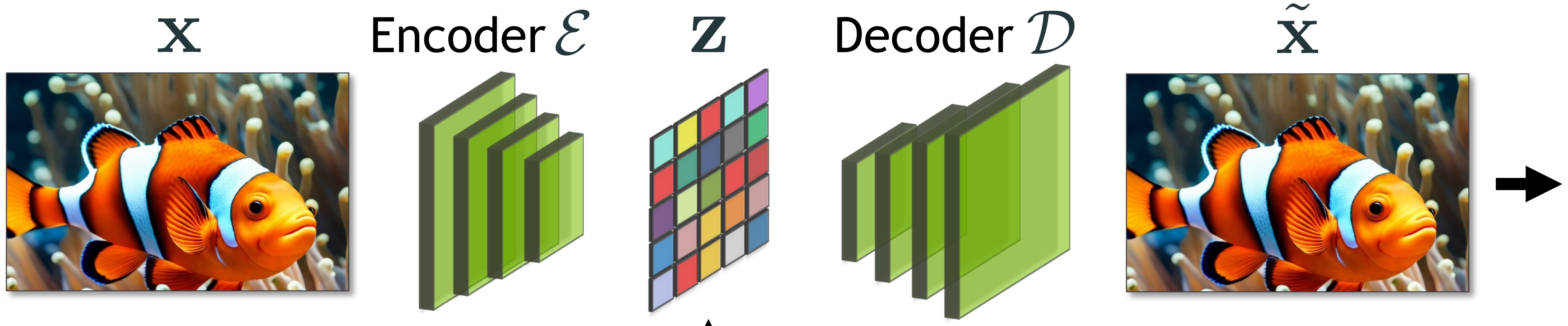
Latent Diffusion Models

Map Data into Compressed Latent Space. Train Diffusion Model efficiently in Latent Space.

- Stage 1:

Train Autoencoder

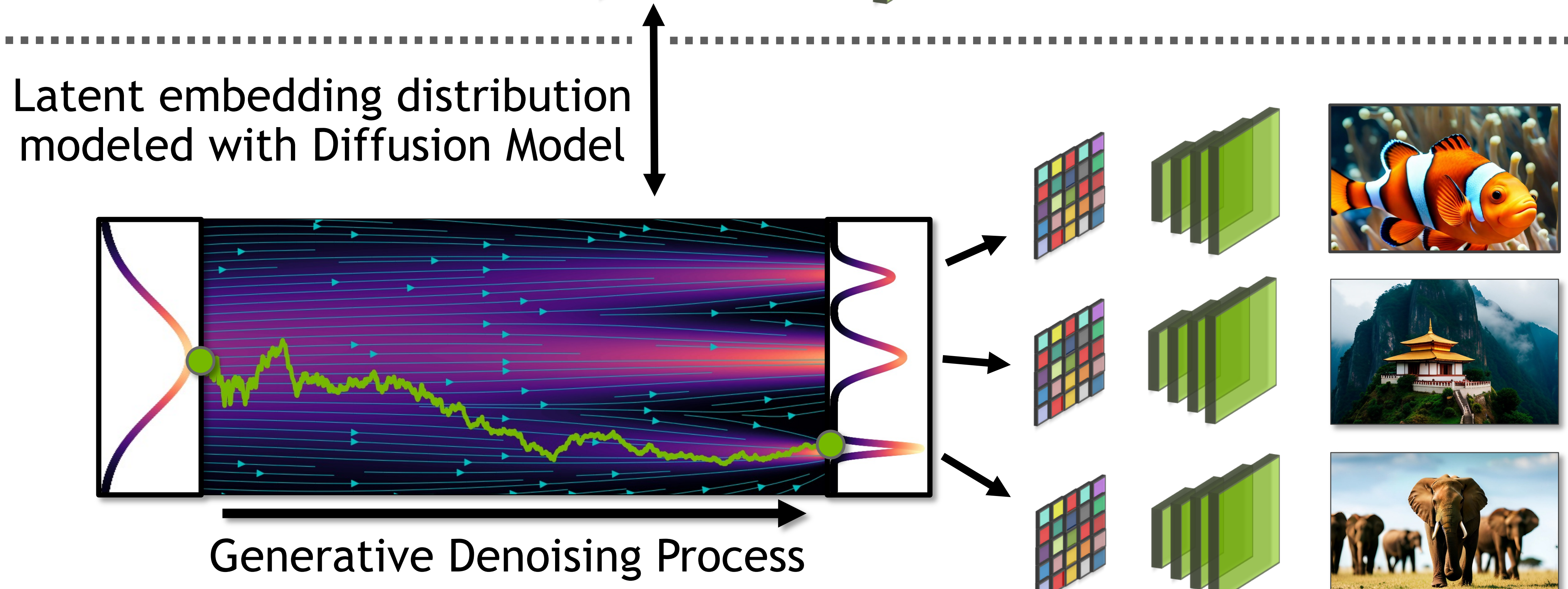
$\tilde{\mathbf{x}} = \mathcal{D}(\mathcal{E}(\mathbf{x}))$



Pixelwise and/or Visual Feature Space (LPIPS) Reconstruction Objective

- Stage 2:

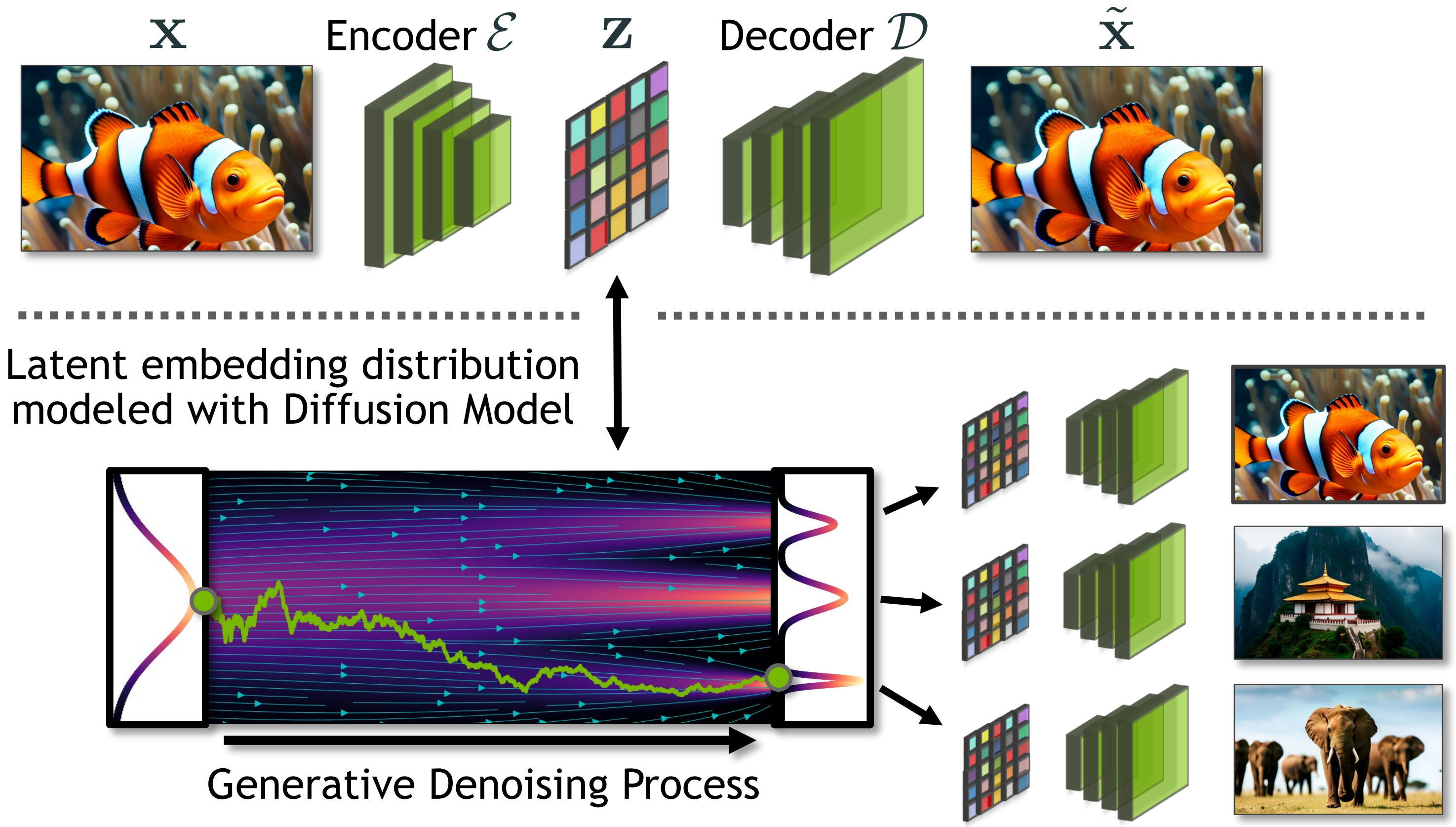
Train **Latent** Diffusion Model



Vahdat et al., "Score-based Generative Modeling in Latent Space", *NeurIPS*, 2021
Rombach et al., "High-Resolution Image Synthesis with Latent Diffusion Models", *CVPR*, 2022
Sinha et al., "D2C: Diffusion-Denoising Models for Few-shot Conditional Generation", *NeurIPS*, 2021
Mittal et al., "Symbolic Music Generation with Diffusion Models", *ISMIR*, 2021

Latent Diffusion Models

Map Data into Compressed Latent Space. Train Diffusion Model efficiently in Latent Space.



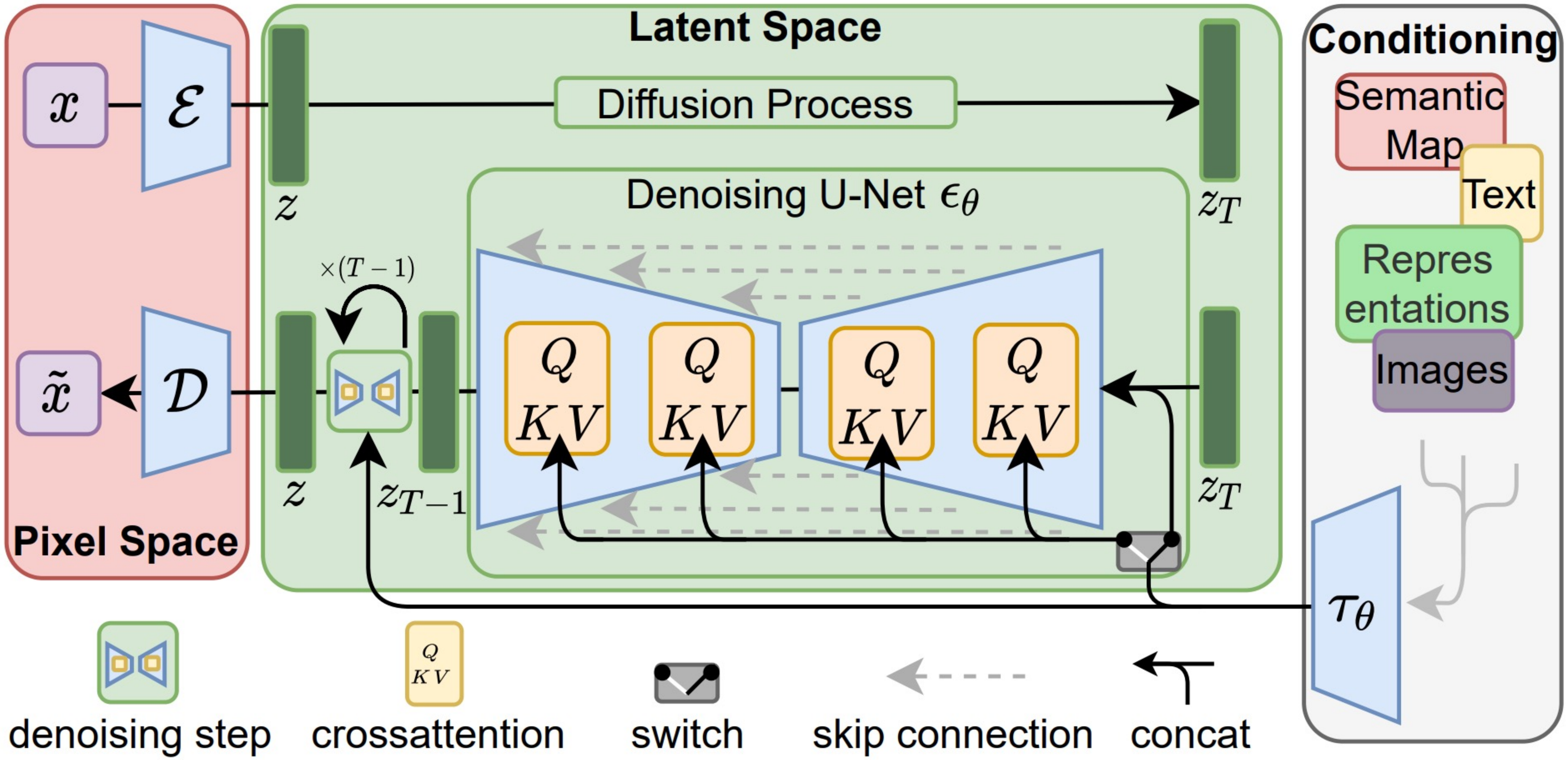
Advantages:

1. *Compressed latent space*: Train diffusion model in **lower resolution** latent space ➡ **computationally more efficiently**
2. *Regularized smooth/compressed latent space*: **Easier task** for diffusion model and **faster sampling**
3. *Flexibility*: **Autoencoder can be tailored to data** (images, video, text, graphs, 3D point clouds, meshes, etc.)

Vahdat et al., "Score-based Generative Modeling in Latent Space", *NeurIPS*, 2021
Rombach et al., "High-Resolution Image Synthesis with Latent Diffusion Models", *CVPR*, 2022
Sinha et al., "D2C: Diffusion-Denoising Models for Few-shot Conditional Generation", *NeurIPS*, 2021
Mittal et al., "Symbolic Music Generation with Diffusion Models", *ISMIR*, 2021

Latent Diffusion Models

Map Data into Compressed Latent Space. Train Diffusion Model efficiently in Latent Space.



$$\tilde{\mathbf{x}} = \mathcal{D}(\mathcal{E}(\mathbf{x}))$$

[Rombach et al., "High-Resolution Image Synthesis with Latent Diffusion Models", *CVPR*, 2022]

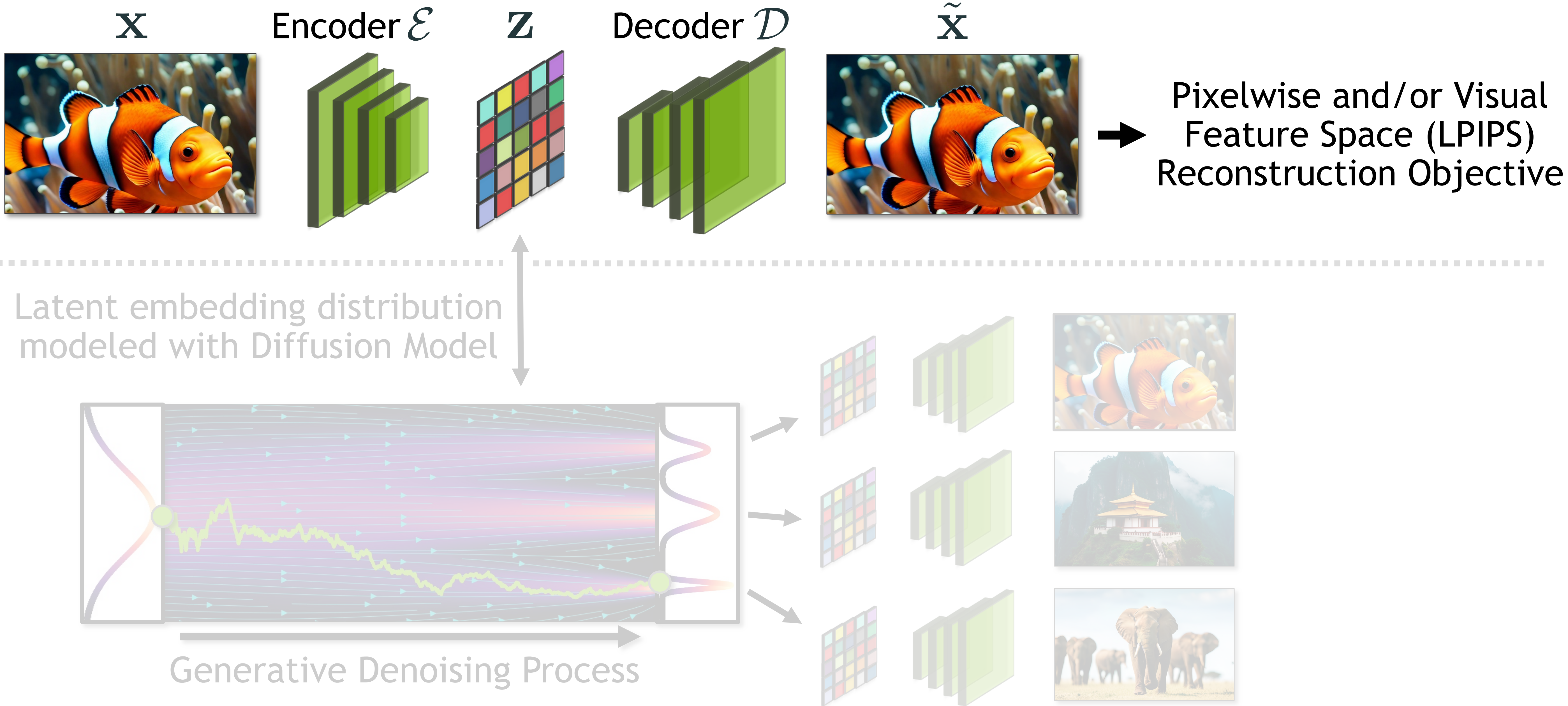
Latent Diffusion Models

Add Adversarial Patch-based Discriminator on top of Reconstruction Loss for Perceptual Compression

- Stage 1:

Train Autoencoder

$\tilde{\mathbf{x}} = \mathcal{D}(\mathcal{E}(\mathbf{x}))$



- Stage 2:

Train Latent Diffusion Model

Rombach et al., "High-Resolution Image Synthesis with Latent Diffusion Models", *CVPR*, 2022

Latent Diffusion Models

Add Adversarial Patch-based Discriminator on top of Reconstruction Loss for Perceptual Compression

- Stage 1:

Train Autoencoder

$\tilde{\mathbf{x}} = \mathcal{D}(\mathcal{E}(\mathbf{x}))$

- Stage 2:

Train Latent Diffusion Model

Rombach et al., "High-Resolution Image Synthesis with Latent Diffusion Models", *CVPR*, 2022

Input

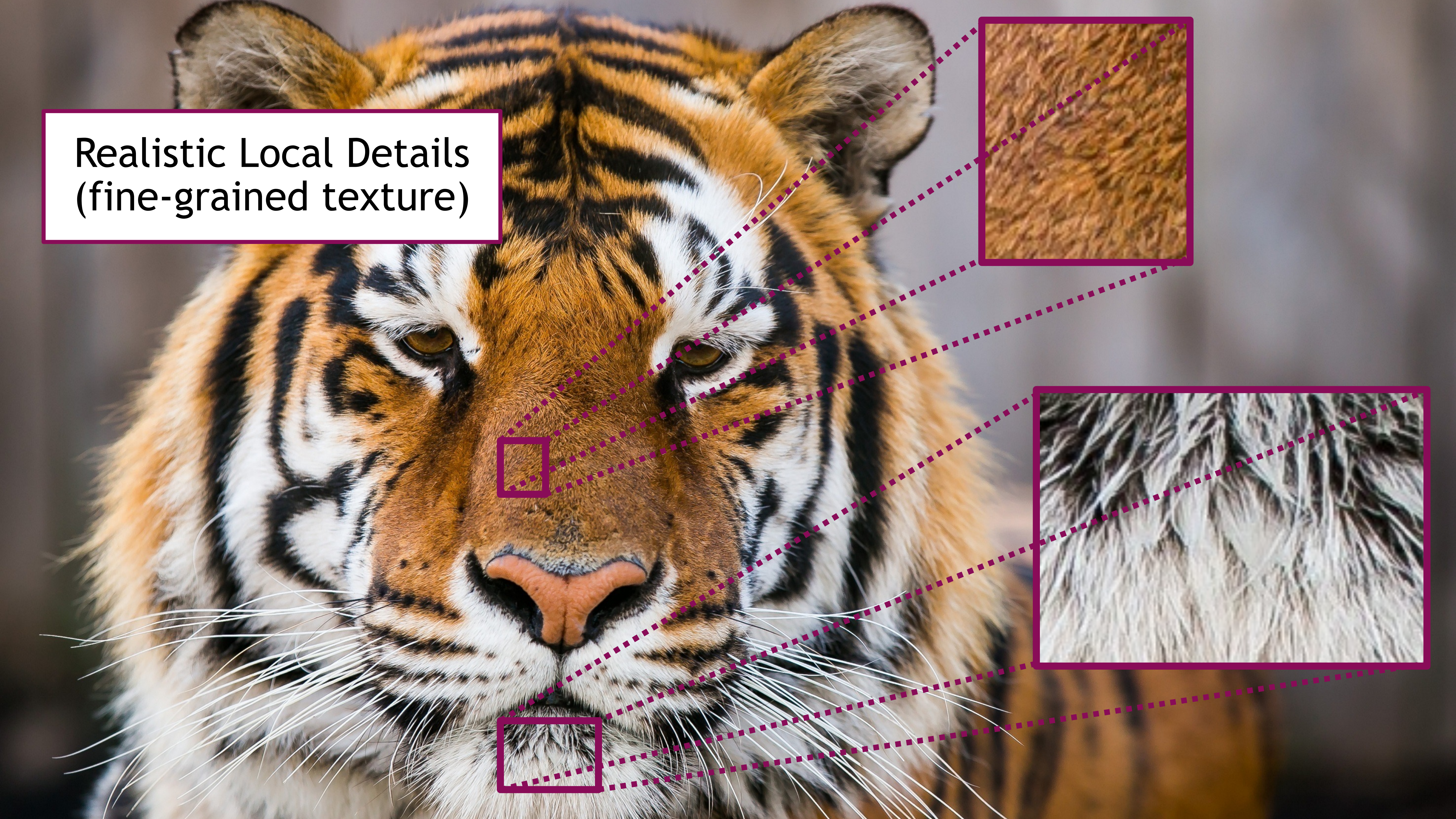
Reconstruction without Discriminator

Reconstruction with Discriminator

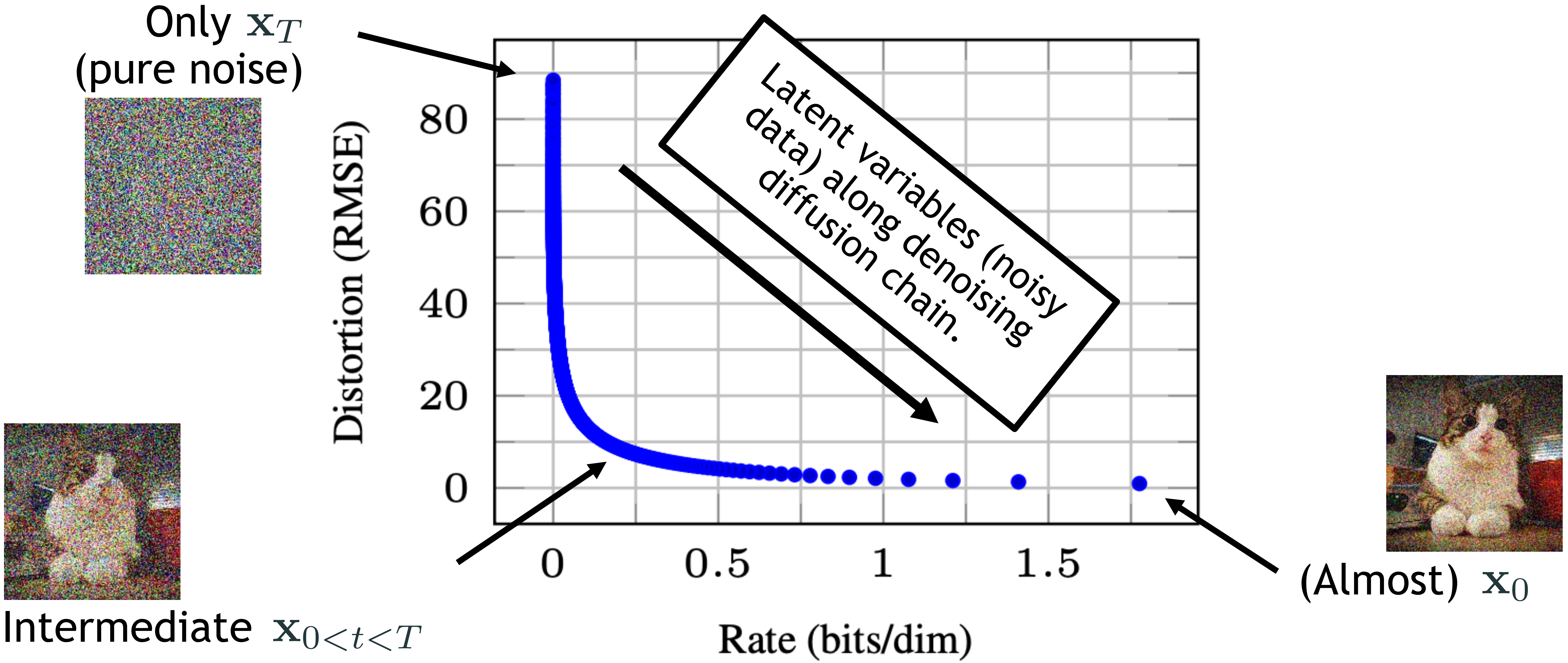
Rombach et al., "High-Resolution Image Synthesis with Latent Diffusion Models", *CVPR*, 2022

What makes an image look realistic and high-quality?


Realistic Global Structure
(correct placement of ears,
eyes, fur pattern, etc.)


Realistic Local Details
(fine-grained texture)

Compression/Encoding in Diffusion Models



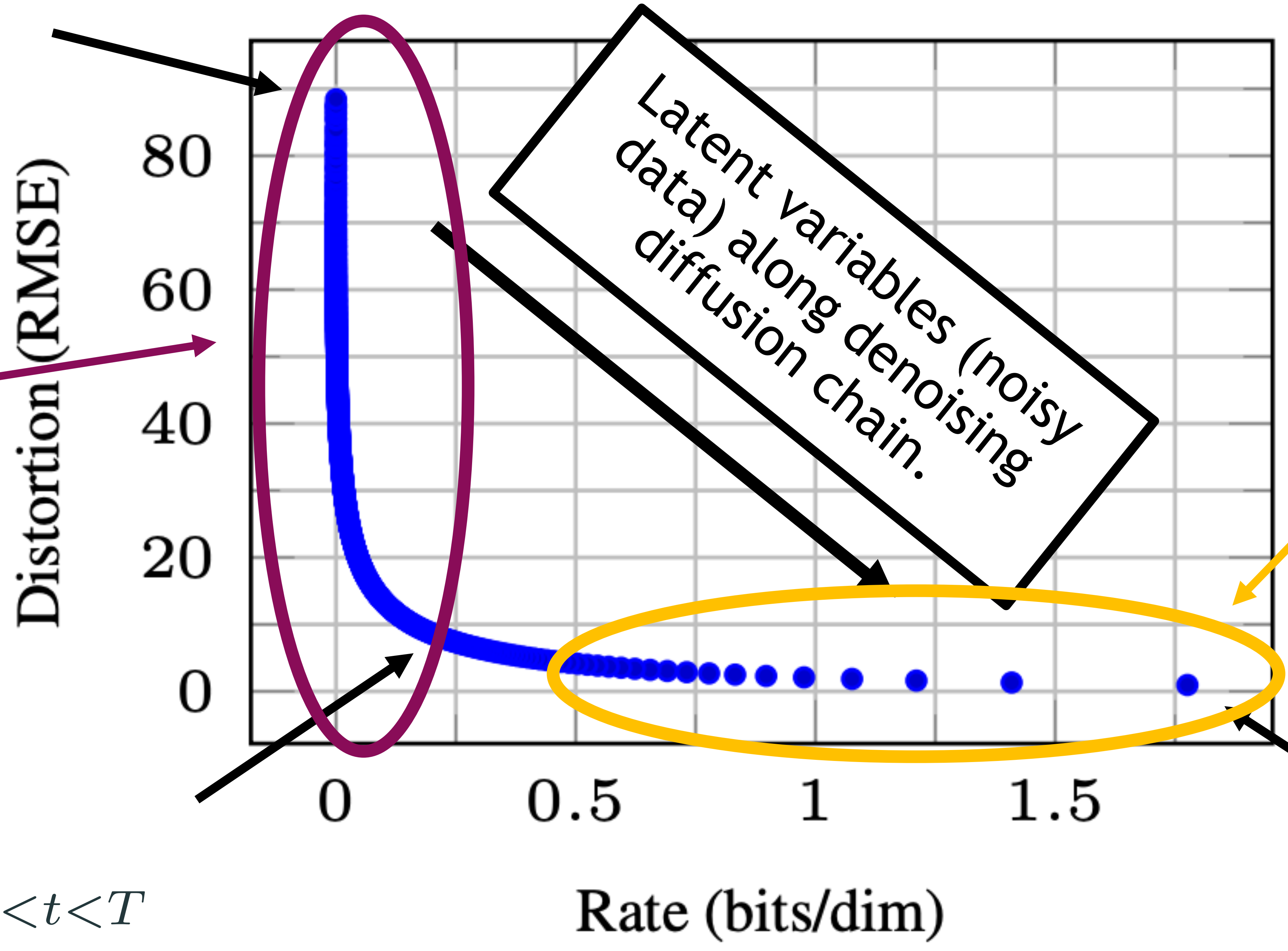
Diffusion models encode images in their noisy latent states.

Ho et al., "Denoising Diffusion Probabilistic Models", *NeurIPS*, 2020

Compression/Encoding in Diffusion Models

Only $\mathbf{x}_T$
(pure noise)

Large-scale
Image Structure



Local, Imperceptible
Details

(Almost) $\mathbf{x}_0$

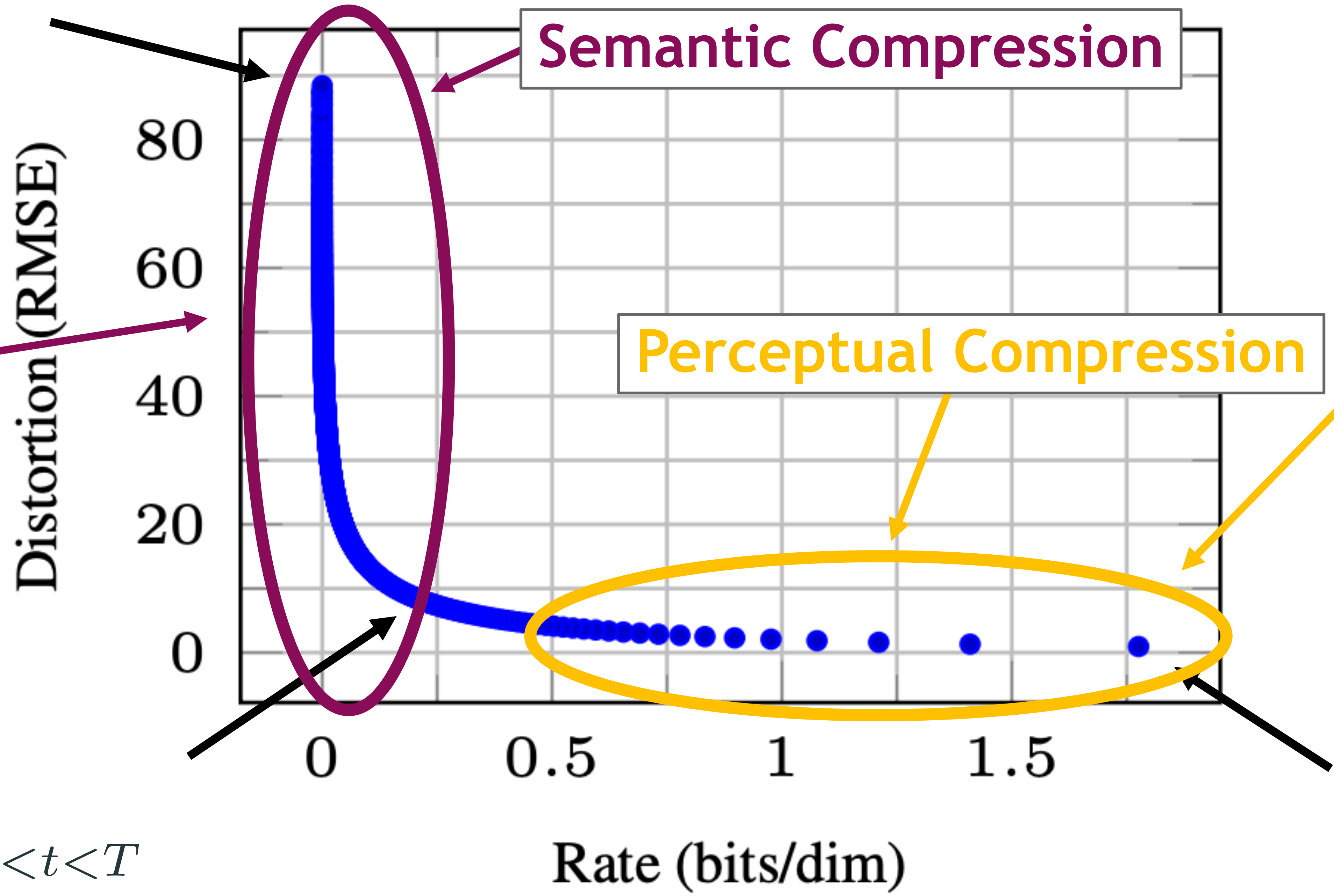
Intermediate $\mathbf{x}_{0<t<T}$

Diffusion models encode images in their noisy latent states.

Ho et al., "Denoising Diffusion Probabilistic Models", *NeurIPS*, 2020

Perceptual and Semantic Compression in Diffusion Models

Large-scale Image Structure



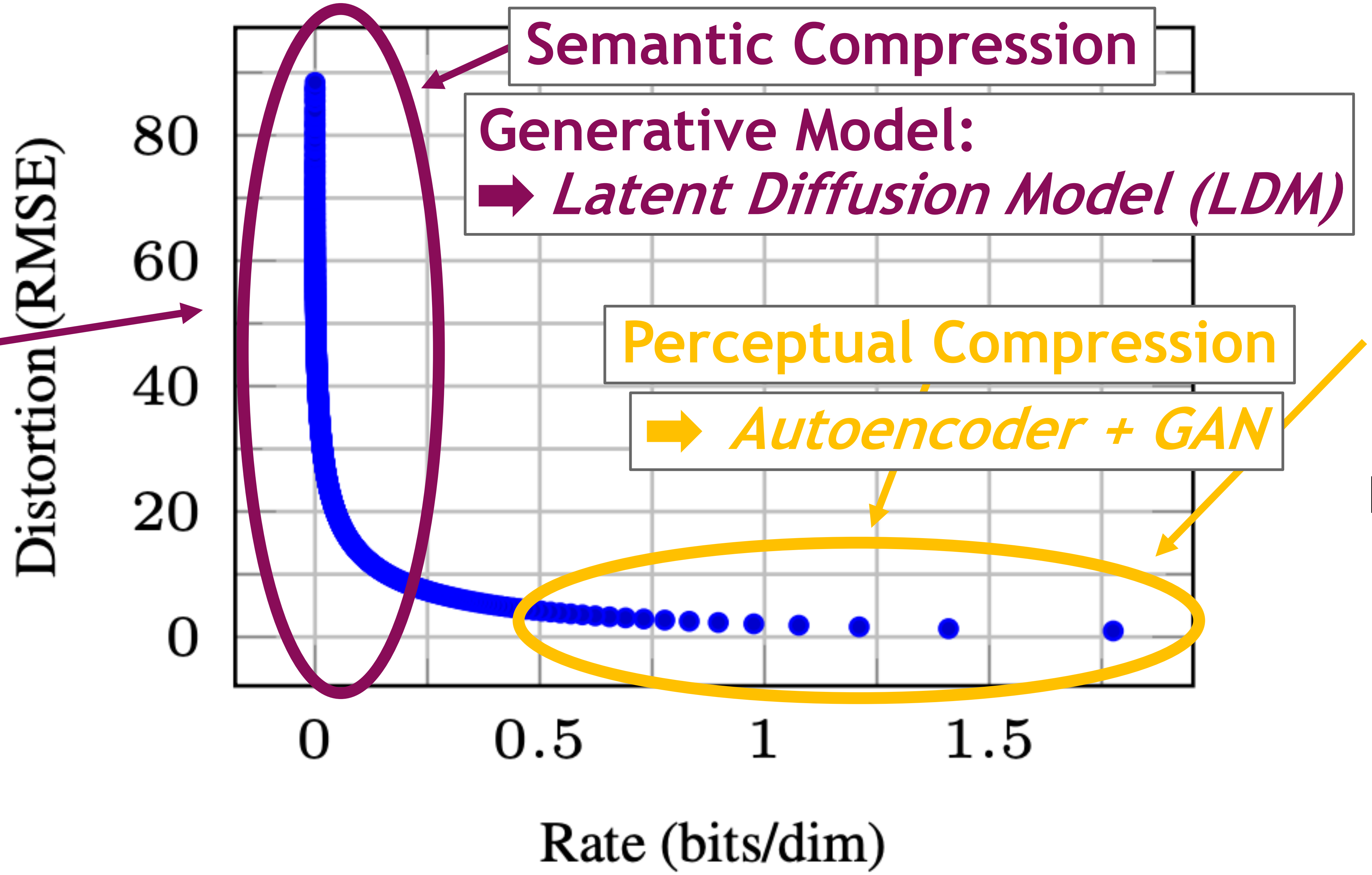
Local, Imperceptible Details

Diffusion models encode images in their noisy latent states.

Ho et al., "Denoising Diffusion Probabilistic Models", *NeurIPS*, 2020

Perceptual and Semantic Compression in *Latent* Diffusion Models

Large-scale
Image Structure



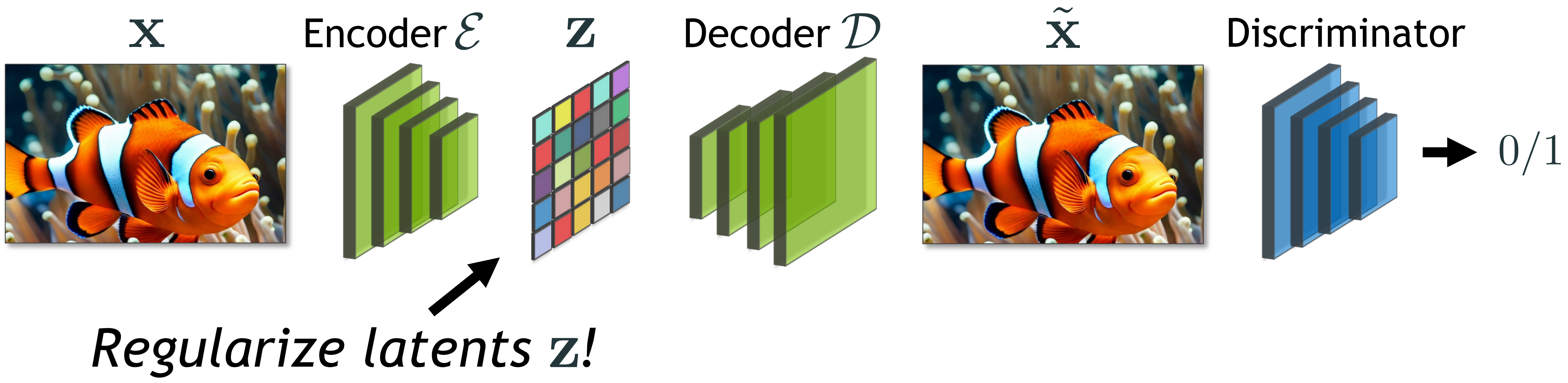
Local, Imperceptible
Details

LDMs: **Latent diffusion model** for **large-scale structure**, **Autoencoder/GAN** for **local details**.

[Rombach et al., "High-Resolution Image Synthesis with Latent Diffusion Models", *CVPR*, 2022]

Latent Space Regularization

Regularize Latent Space for better Compression and easier Training of Latent Space Diffusion Models



- **Option 1: Kullback-Leibler (KL) regularization**

  Parametrize encoder by diagonal Gaussian, regularize towards standard normal distribution, as in regular VAEs.

  Use very small weight for KL regularization term (weak regularization).

Encoder distribution:

$$q_{\mathcal{E}}(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \mathcal{E}_\mu, \mathcal{E}_\sigma^2)$$

KL regularization in latent space:

$$\mathrm{KL}\left(q_{\mathcal{E}}(\mathbf{z}|\mathbf{x}) || \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})\right)$$

Rombach et al., "High-Resolution Image Synthesis with Latent Diffusion Models", *CVPR*, 2022

Details on auto-encoder training

The auto-encoder is trained as a **VAE+GAN**

- $\mathcal{E}$: (Stochastic) encoder, $\mathcal{E}(\boldsymbol{x}) = (\mathcal{E}_{\boldsymbol{\mu}}(\boldsymbol{x}), \mathcal{E}_{\boldsymbol{\sigma}}(\boldsymbol{x}))$
- $\mathcal{D}$: (Stochastic) decoder
- $\mathbf{D}$: Patch-based discriminator (like for SRGAN, pix2pix,...)

$$\min_{\mathcal{E},\mathcal{D}} \max_{\mathbf{D}} L_{\text{Autoencoder}}(\boldsymbol{x})$$

$$L_{\text{Autoencoder}}(\boldsymbol{x}) = L_{\text{reg}}(\boldsymbol{x}, \mathcal{E}, \mathcal{D}) + L_{\text{rec}}(\boldsymbol{x}, \mathcal{D}(\mathcal{E}(\boldsymbol{x}))) + L_{\text{adv}}(\mathbf{D}, \boldsymbol{x}, \mathcal{D}(\mathcal{E}(\boldsymbol{x})))$$

where

$$\begin{aligned} L_{\text{reg}}(\boldsymbol{x}, \mathcal{E}, \mathcal{D}) &= \lambda D_{\text{KL}}(\mathcal{N}(\mathcal{E}_{\boldsymbol{\mu}}(\boldsymbol{x}), \operatorname{diag}(\mathcal{E}_{\boldsymbol{\sigma}}(\boldsymbol{x})^2)) || \mathcal{N}(\mathbf{0}, I_d)) \\ &= \frac{\lambda}{2} \sum_{j=1}^{k} \left(\mu_j(\boldsymbol{x})^2 + \sigma_j(\boldsymbol{x})^2 - 1 - \log \sigma_j(\boldsymbol{x})^2 \right) \end{aligned}$$

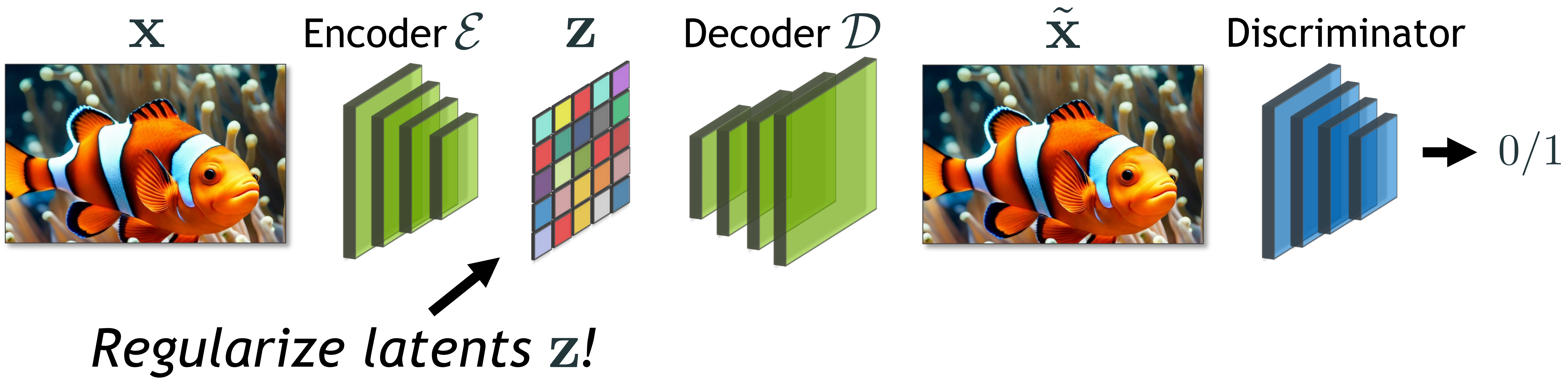
$$\boldsymbol{x}_{\text{rec}} = \mathcal{D}(z) \quad \text{with} \quad z = \mathcal{E}_{\boldsymbol{\mu}}(\boldsymbol{x}) + \mathcal{E}_{\boldsymbol{\sigma}}(\boldsymbol{x}) \odot \boldsymbol{\varepsilon}, \quad \boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, I_d).$$

$$L_{\text{rec}}(\boldsymbol{x}, \mathcal{D}(\mathcal{E}(\boldsymbol{x}))) = \|\boldsymbol{x} - \boldsymbol{x}_{\text{rec}}\|^2$$

$$L_{\text{adv}}(\mathbf{D}, \boldsymbol{x}, \mathcal{D}(\mathcal{E}(\boldsymbol{x}))) = \log(\mathbf{D}(\boldsymbol{x})) + \log(1 - \mathbf{D}(\boldsymbol{x}_{\text{rec}}))$$

Latent Space Regularization

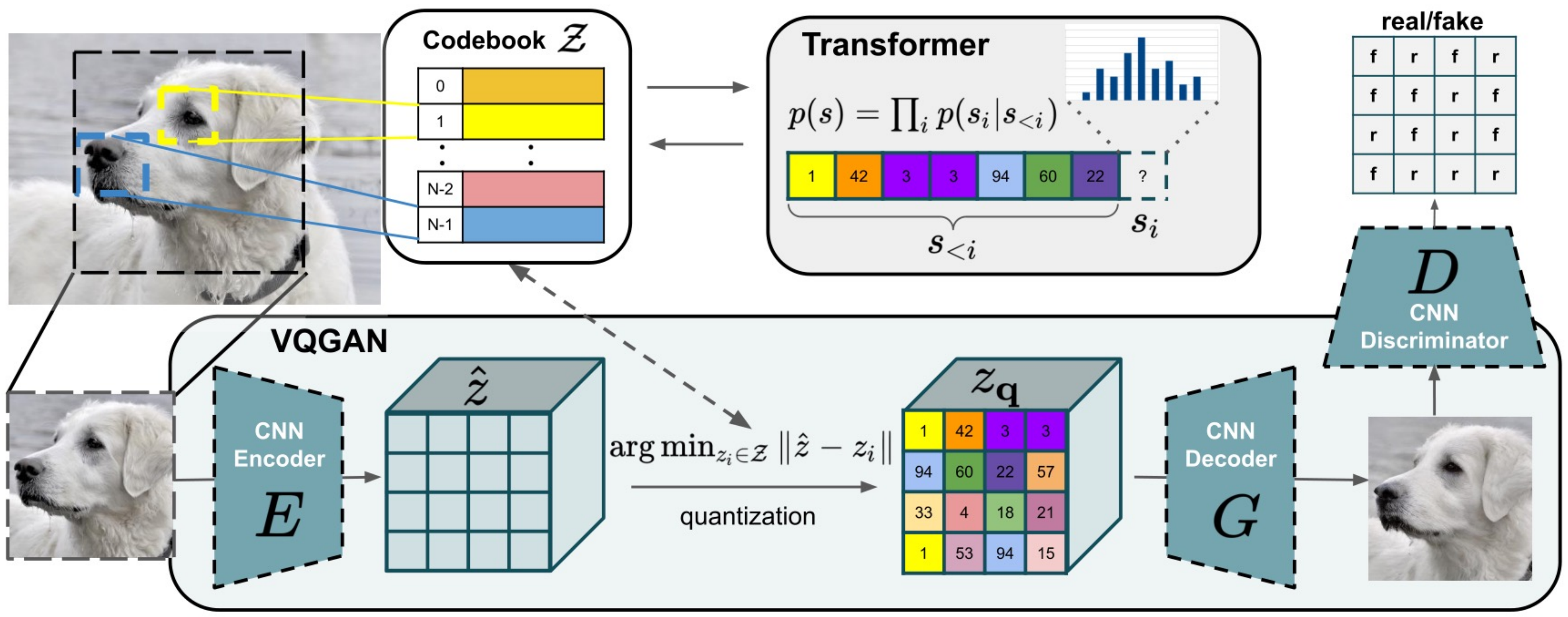
Regularize Latent Space for better Compression and easier Training of Latent Space Diffusion Models



- **Option 2: Vector Quantization (VQ) regularization**

  Discretize latent encodings using finite-sized learnable codebook as in VQ-VAEs (implemented by vector-quantization layer in decoder).

  Use large codebook size (weak regularization).



Esser et al., "*Taming Transformers for High-Resolution Image Synthesis*", CVPR, 2021

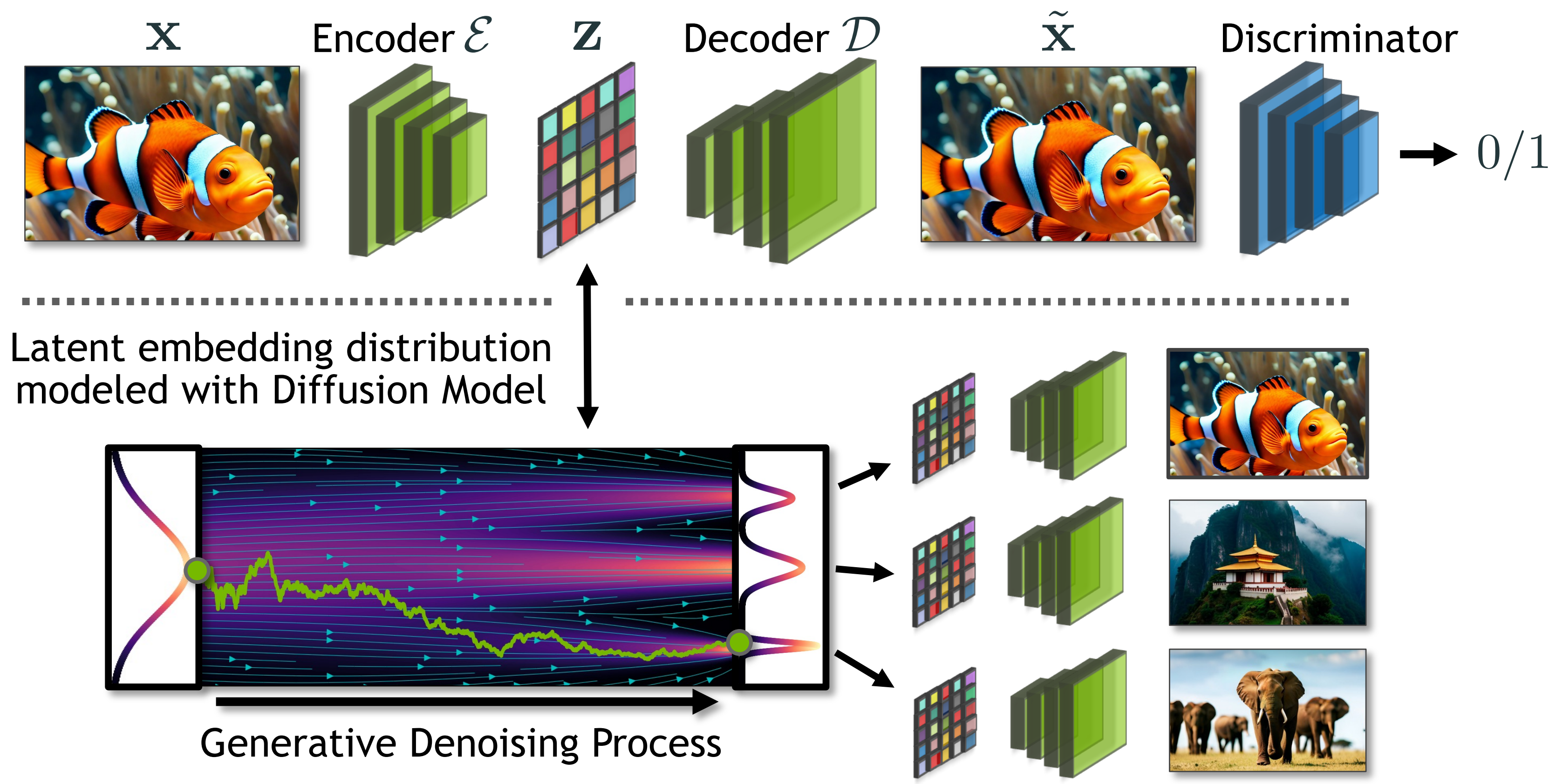
Latent Diffusion Models

Latent Diffusion Models offer Excellent Trade-off between Performance and Compute Demands

LDM "Recipe":

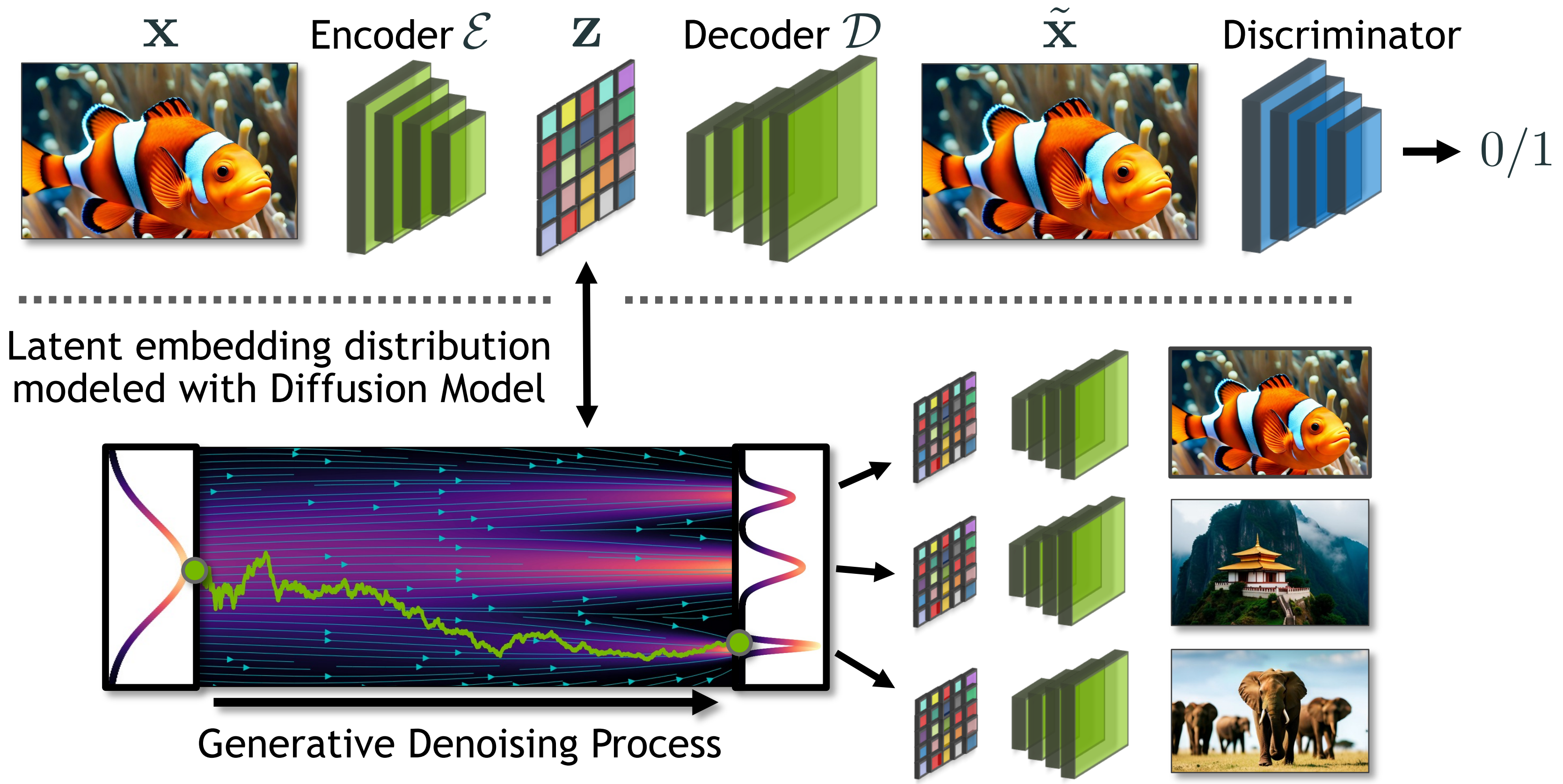
1. Train strong autoencoder
   - Compress…
     (downsampling factor / latent space regularization)
   - …while ensuring high visual quality on reconstructions
     ("upper bound" on synthesis quality)

2. Train efficient latent diffusion model
   - Latent space compression/regularization makes diffusion model training easier ➡ but trade-off with respect quality?
   - Discriminator ➡ high quality despite compression (re-generate details, not encode)!



Latent Diffusion Models

Latent Diffusion Models offer Excellent Trade-off between Performance and Compute Demands



➡ LDM with appropriate regularization, compression, downsampling ratio and strong autoencoder reconstruction:

- **Computationally efficient** diffusion model in latent space (compression & lower resolution).
- Yet **very high-performance** (latent diffusion + autoencoder + discriminator = ❤).
- **Highly flexible** (can adjust autoencoder for different tasks and data).

Image Generation with Latent Diffusion Models

Many state-of-the-art large-scale text-to-image models are latent diffusion models:

- Stability AI's **Stable Diffusion**
- Meta's **Emu**
- OpenAI's **Dall-E 3**?

Common observation:

- Latent diffusion model **technology is mature** for practical image generation.
- The above models all achieve their high-performance by **sophisticated data captioning** and **filtering** and **fine-tuning** strategies.

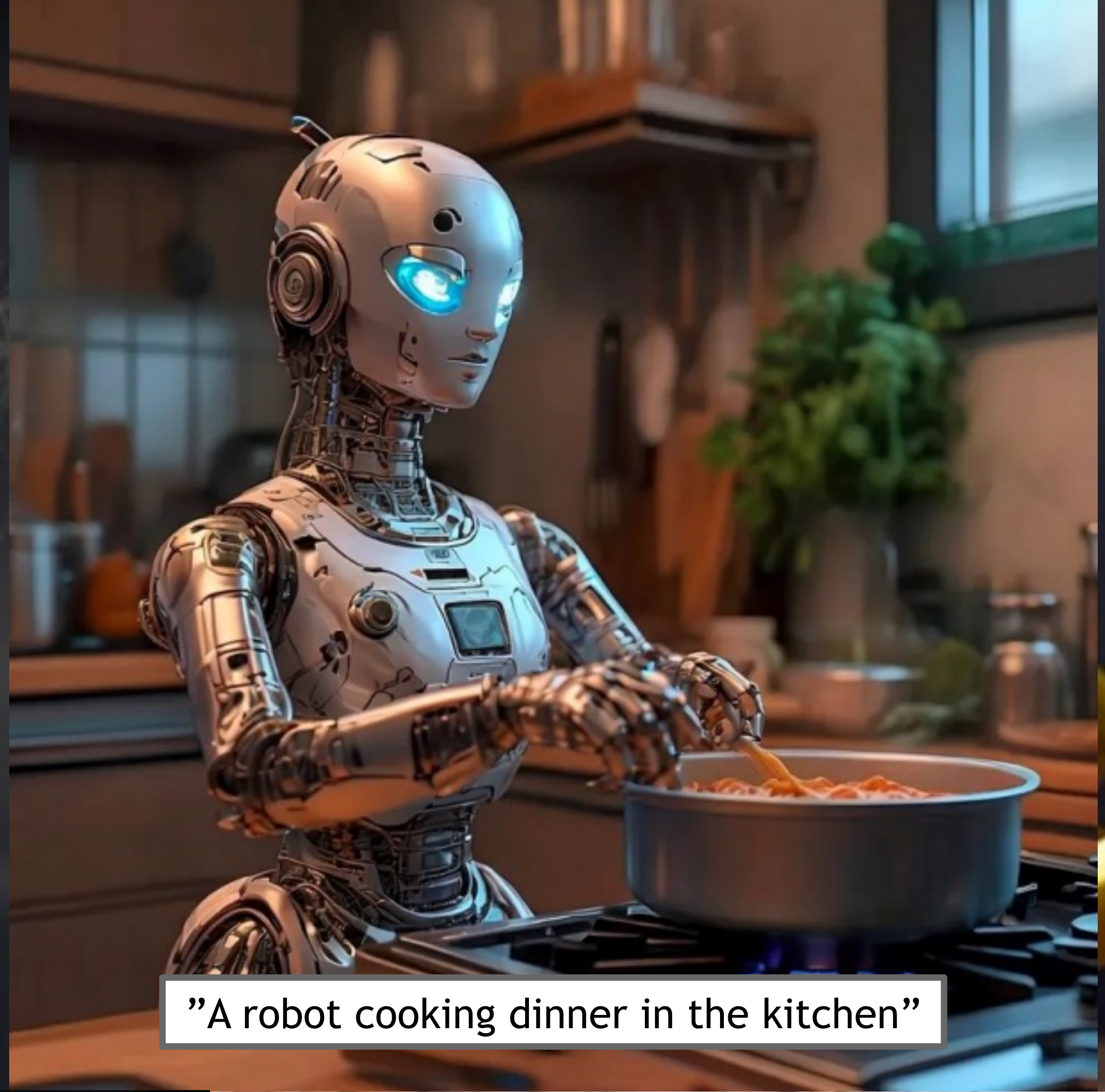
Rombach et al., "High-Resolution Image Synthesis with Latent Diffusion Models", *CVPR*, 2022
Dai et al., "Emu: Enhancing Image Generation Models Using Photogenic Needles in a Haystack", *arXiv*, 2023
Betker et al., "Improving Image Generation with Better Captions" *(DALL-E 3), 2023*

https://stability.ai/news/stable-diffusion-sdxl-1-announcement

https://stability.ai/news/stable-diffusion-sdxl-1-announcement

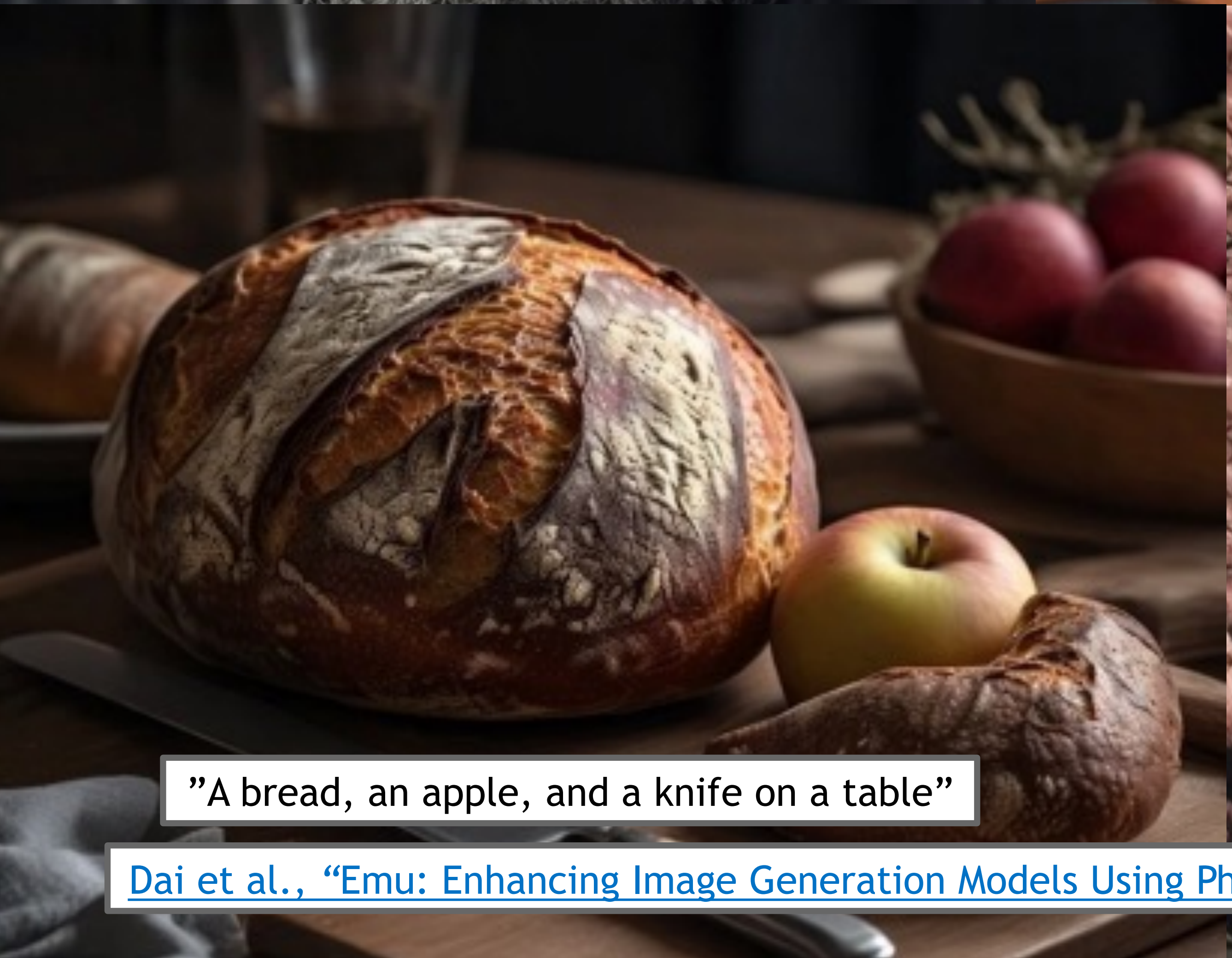
https://stability.ai/news/stable-diffusion-sdxl-1-announcement

"An old man with green eyes and a long grey beard"

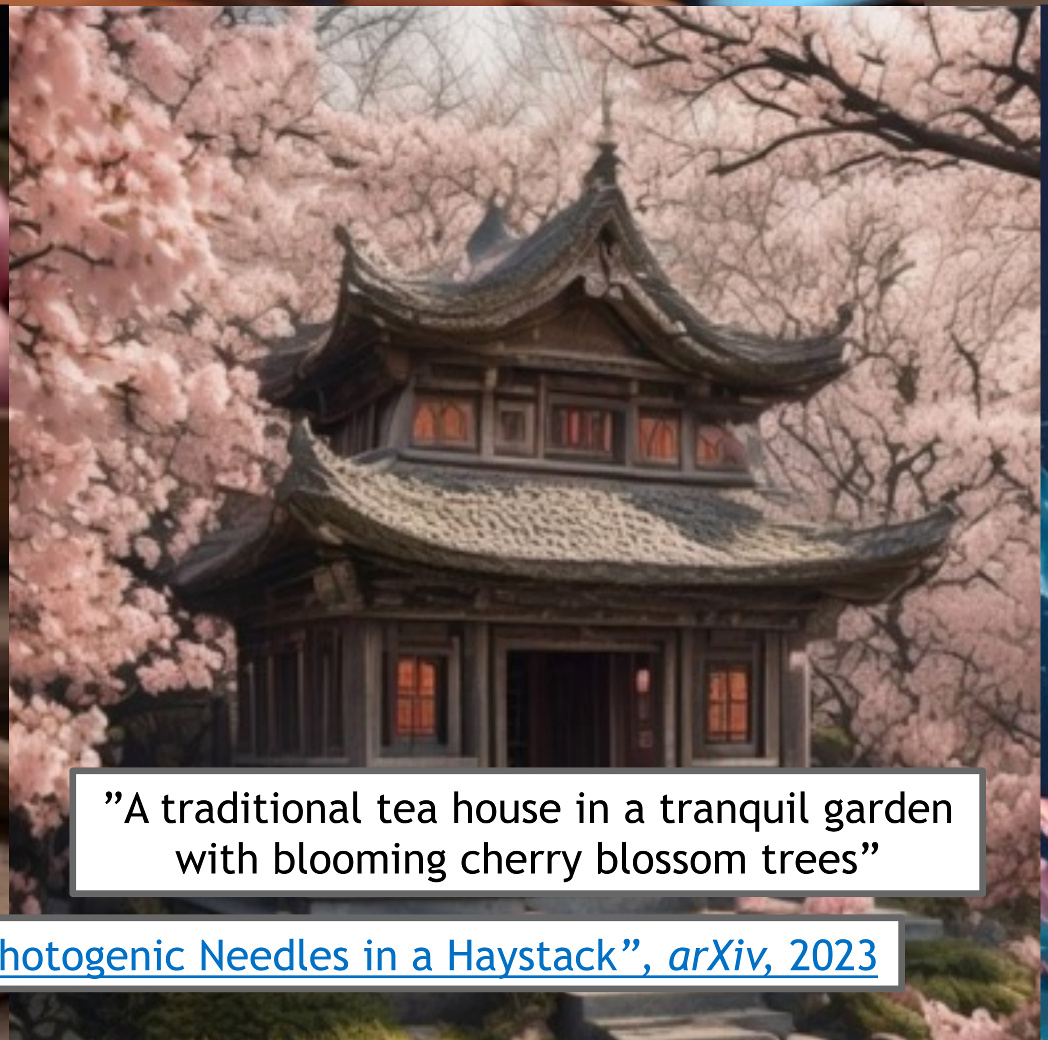


"A robot cooking dinner in the kitchen"

"A cool orange cat wearing sunglasses playing a guitar with a group of dancing bananas"



"A bread, an apple, and a knife on a table"



"A traditional tea house in a tranquil garden with blooming cherry blossom trees"

"A painting of an adorable rabbit sitting on a colorful splash"

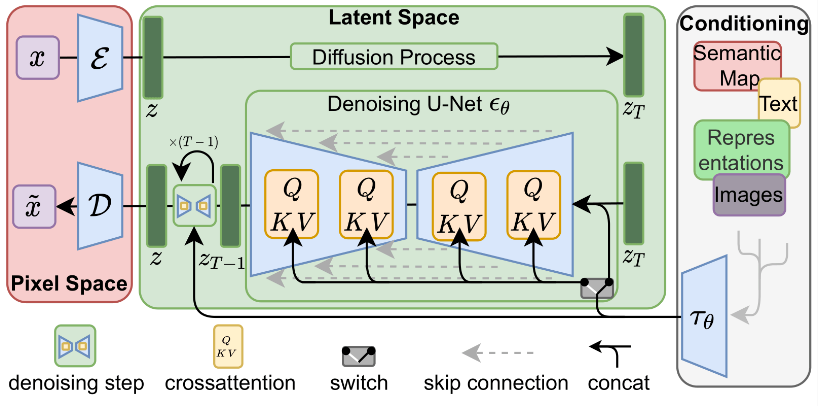
Dai et al., "Emu: Enhancing Image Generation Models Using Photogenic Needles in a Haystack", *arXiv*, 2023

[Dai et al., "Emu: Enhancing Image Generation Models Using Photogenic Needles in a Haystack", *arXiv*, 2023]()

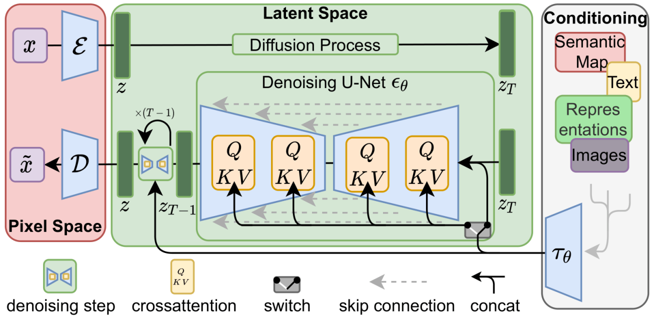
Betker et al., “Improving Image Generation with Better Captions” *(DALL-E 3), 2023*

Betker et al., "Improving Image Generation with Better Captions" *(DALL-E 3), 2023*

More technical details



- Diffusion sampling done with the DDIM deterministic approach with 200 steps (but trained as a DDPM with $T = 1000$ steps).



Text-to-image synthesis:

- Text is "tokenized" using BERT (Devlin et al., 2019) and then pass through a transformer τ_θ.
- Text conditioning using **cross-attention**: Replace the Unet self-attention layer with a shallow (unmasked) transformer consisting of blocks with alternating layers of (i) self-attention, (ii) a position-wise MLP and (iii) a cross-attention layer.
- Trained on LAION-400M database (text and image) (or LAION-5B...).

Controlling latent diffusion models

Introducing a specific object into generated images

DreamBooth: Fine Tuning Text-to-Image Diffusion Models for Subject-Driven Generation

Nataniel Ruiz[*,1,2] Yuanzhen Li[1] Varun Jampani[1]
Yael Pritch[1] Michael Rubinstein[1] Kfir Aberman[1]
[1] Google Research [2] Boston University

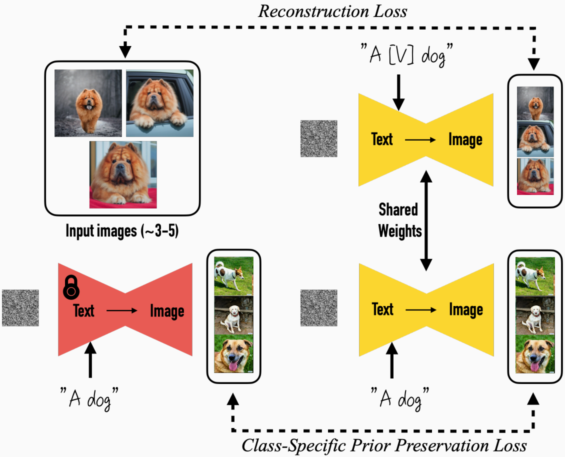
Figure 1. With just a few images (typically 3-5) of a subject (left), *DreamBooth*—our AI-powered photo booth—can generate a myriad of images of the subject in different contexts (right), using the guidance of a text prompt. The results exhibit natural interactions with the environment, as well as novel articulations and variation in lighting conditions, all while maintaining high fidelity to the key visual features of the subject.

(source: (Ruiz et al., 2023))

Introducing a specific object into generated images

(Ruiz et al., 2023)

"**Fine-tuning:** Given 3-5 images of a subject, we fine-tune a text-to-image diffusion model with the input images paired with a text prompt containing a unique identifier and the name of the class the subject belongs to (e.g., "A [V] dog")"



"in parallel, we apply a class-specific prior preservation loss, which leverages the semantic prior that the model has on the class and encourages it to generate diverse instances belong to the subject's class using the class name in a text prompt (e.g., "A dog")."

Better spatial control for compositioning

Adding Conditional Control to Text-to-Image Diffusion Models

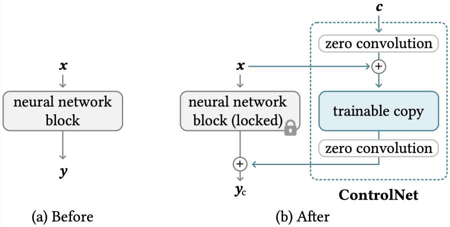
Lvmin Zhang, Anyi Rao, and Maneesh Agrawala
Stanford University
{lvmin, anyirao, maneesh}@cs.stanford.edu

Figure 1: Controlling Stable Diffusion with learned conditions. ControlNet allows users to add conditions like Canny edges (top), human pose (bottom), *etc*., to control the image generation of large pretrained diffusion models. The default results use the prompt "a high-quality, detailed, and professional image". Users can optionally give prompts like the "chef in kitchen".

(source: ControlNet (Zhang et al., 2023))

ControlNet (Zhang et al., 2023)

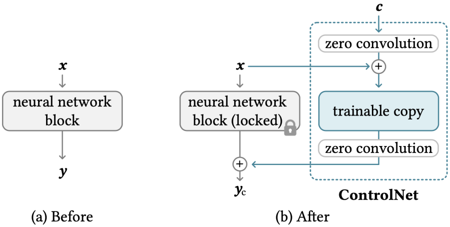
The simple idea:



- Add a copy of the block with "zero-convolution" = 1×1 convolution blocks initiated by zero.
- Apply end-to-end training on medium size dataset to the trainable copy.
- Avoid "catastrophic forgetting" of the original stable diffusion mdoel.

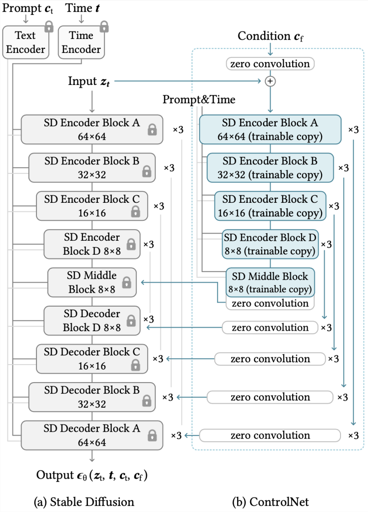
ControlNet (Zhang et al., 2023)

The simple idea:



- Add a copy of the block with "zero-convolution" = 1×1 convolution blocks initiated by zero.
- Apply end-to-end training on medium size dataset to the trainable copy.
- Avoid "catastrophic forgetting" of the original stable diffusion mdoel.

The implementation:



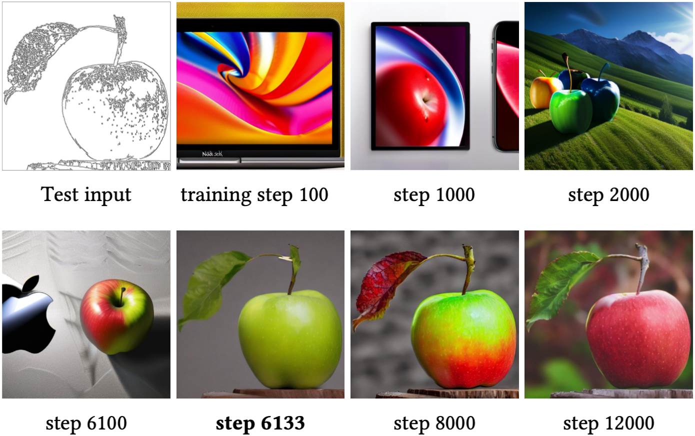


Figure 4: The sudden convergence phenomenon. Due to the zero convolutions, ControlNet always predicts high-quality images during the entire training. At a certain step in the training process (*e.g.*, the 6133 steps marked in bold), the model suddenly learns to follow the input condition.

ControlNet (Zhang et al., 2023)

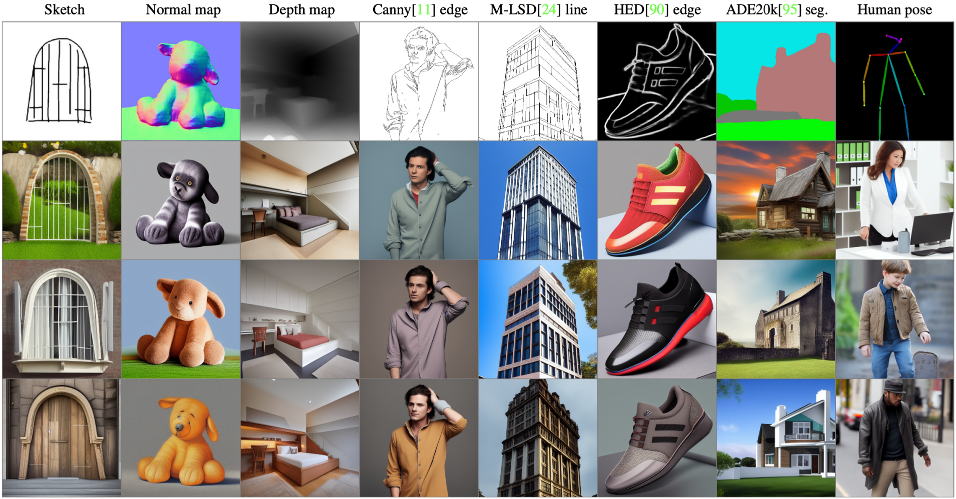


Figure 7: Controlling Stable Diffusion with various conditions **without prompts**. The top row is input conditions, while all other rows are outputs. We use the empty string as input prompts. All models are trained with general-domain data. The model has to recognize semantic contents in the input condition images to generate images.

Diffusion models:

- (Latent) diffusion models have become a mature framework for generative modeling.
- They are large models that require very long training with very large datasets...
- But once trained they can be used as generic image priors.

Latent diffusion models:

- Latent diffusion models allow for new image editing/generation techniques.
- Bridge between NLP and image modeling.
- Latent diffusion models are booming for video generation from text.
- Can be applied to any modality with some structured latent space, but not SOTA for all modalities (e.g. not competitive for text generation).

References

References

Anderson, B. D. (1982). Reverse-time diffusion equation models. *Stochastic Processes and their Applications*, 12(3):313–326.

Chung, H., Kim, J., Mccann, M. T., Klasky, M. L., and Ye, J. C. (2023). Diffusion posterior sampling for general noisy inverse problems. In *The Eleventh International Conference on Learning Representations*.

De Bortoli, V. (2022). Convergence of denoising diffusion models under the manifold hypothesis. *Transactions on Machine Learning Research*. Expert Certification.

Devlin, J., Chang, M., Lee, K., and Toutanova, K. (2019). BERT: pre-training of deep bidirectional transformers for language understanding. In Burstein, J., Doran, C., and Solorio, T., editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics.

Dhariwal, P. and Nichol, A. Q. (2021). Diffusion models beat GANs on image synthesis. In Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems*.

Haussmann, U. G. and Pardoux, E. (1986). Time reversal of diffusions. *The Annals of Probability*, 14(4):1188 – 1205.

Ho, J., Jain, A., and Abbeel, P. (2020). Denoising diffusion probabilistic models. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 6840–6851. Curran Associates, Inc.

Karras, T., Aittala, M., Aila, T., and Laine, S. (2022). Elucidating the design space of diffusion-based generative models. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K., editors, *Advances in Neural Information Processing Systems*.

Kawar, B., Elad, M., Ermon, S., and Song, J. (2022). Denoising diffusion restoration models. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A., editors, *Advances in Neural Information Processing Systems*, volume 35, pages 23593–23606. Curran Associates, Inc.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In Bengio, Y. and LeCun, Y., editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Kingma, D. P. and Gao, R. (2023). Understanding diffusion objectives as the ELBO with simple data augmentation. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Lugmayr, A., Danelljan, M., Romero, A., Yu, F., Timofte, R., and Van Gool, L. (2022). Repaint: Inpainting using denoising diffusion probabilistic models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11461–11471.

Nichol, A. Q. and Dhariwal, P. (2021). Improved denoising diffusion probabilistic models. In Meila, M. and Zhang, T., editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 8162–8171. PMLR.

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. (2022). High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10684–10695.

Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In Navab, N., Hornegger, J., Wells, W. M., and Frangi, A. F., editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, Cham. Springer International Publishing.

Ruiz, N., Li, Y., Jampani, V., Pritch, Y., Rubinstein, M., and Aberman, K. (2023). Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 22500–22510.

Saharia, C., Chan, W., Saxena, S., Li, L., Whang, J., Denton, E., Ghasemipour, S. K. S., Ayan, B. K., Mahdavi, S. S., Lopes, R. G., Salimans, T., Ho, J., Fleet, D. J., and Norouzi, M. (2022). Photorealistic text-to-image diffusion models with deep language understanding.

Saharia, C., Ho, J., Chan, W., Salimans, T., Fleet, D. J., and Norouzi, M. (2023). Image super-resolution via iterative refinement. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(4):4713–4726.

Song, J., Meng, C., and Ermon, S. (2021a). Denoising diffusion implicit models. In *International Conference on Learning Representations*.

Song, J., Vahdat, A., Mardani, M., and Kautz, J. (2023). Pseudoinverse-guided diffusion models for inverse problems. In *International Conference on Learning Representations*.

Song, Y. and Ermon, S. (2020). Improved techniques for training score-based generative models. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 12438–12448. Curran Associates, Inc.

Song, Y., Sohl-Dickstein, J., Kingma, D. P., Kumar, A., Ermon, S., and Poole, B. (2021b). Score-based generative modeling through stochastic differential equations. In *9th International Conference on Learning Representations, ICLR 2021*. OpenReview.net.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. (2017). Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Zhang, L., Rao, A., and Agrawala, M. (2023). Adding conditional control to text-to-image diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 3836–3847.