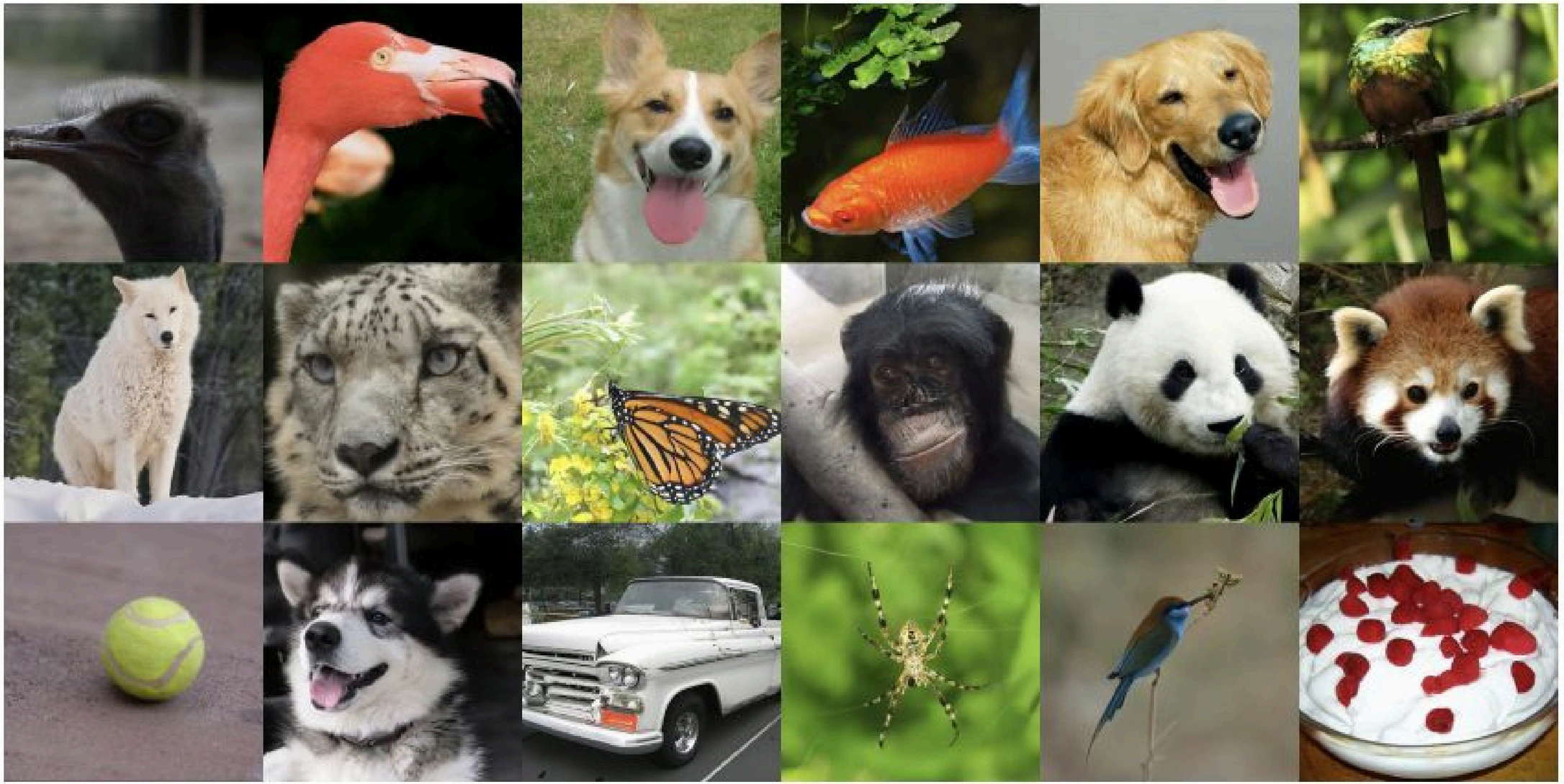
INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY

HYDERABAD

Introduction to Diffusion Models

Anant Garg



Diffusion Models Beat GANs on Image Synthesis [Dhariwal et. al., 2022]

“Oil painting of Taj Mahal in the style of Van Gogh”

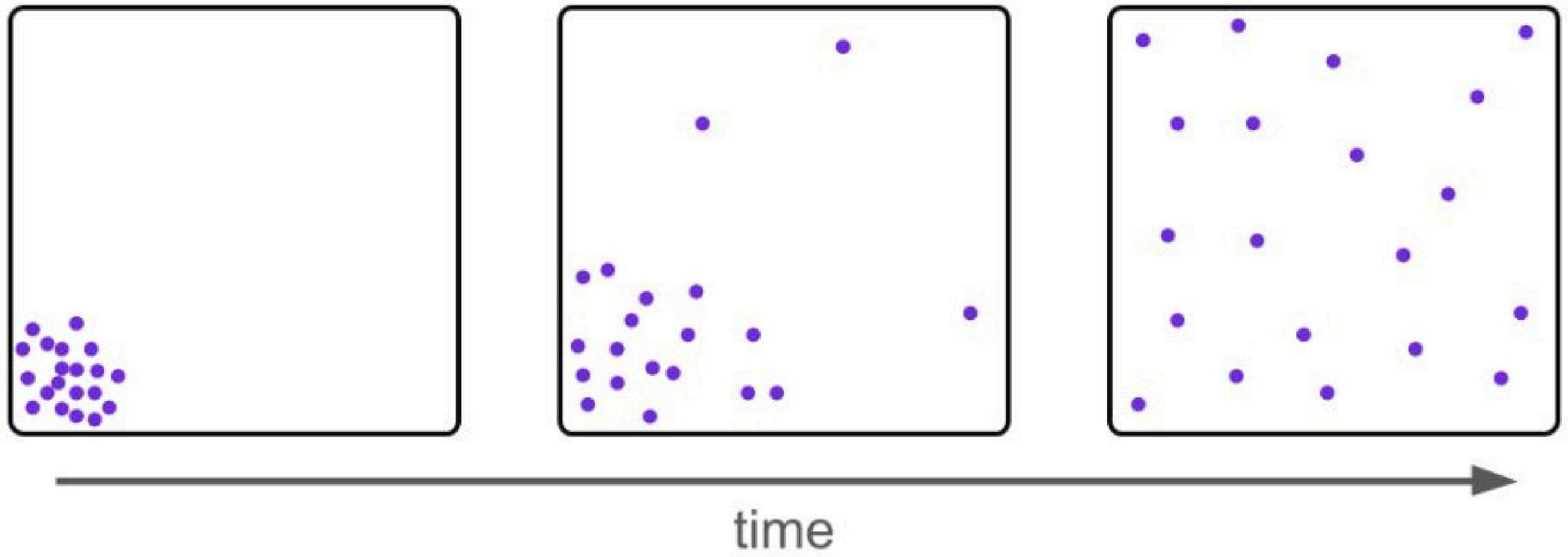
[OpenAI’s DALLE-2]

“Mona Lisa as a thug life character, smoking cigar, wearing shades and a locket with dollar symbol.”

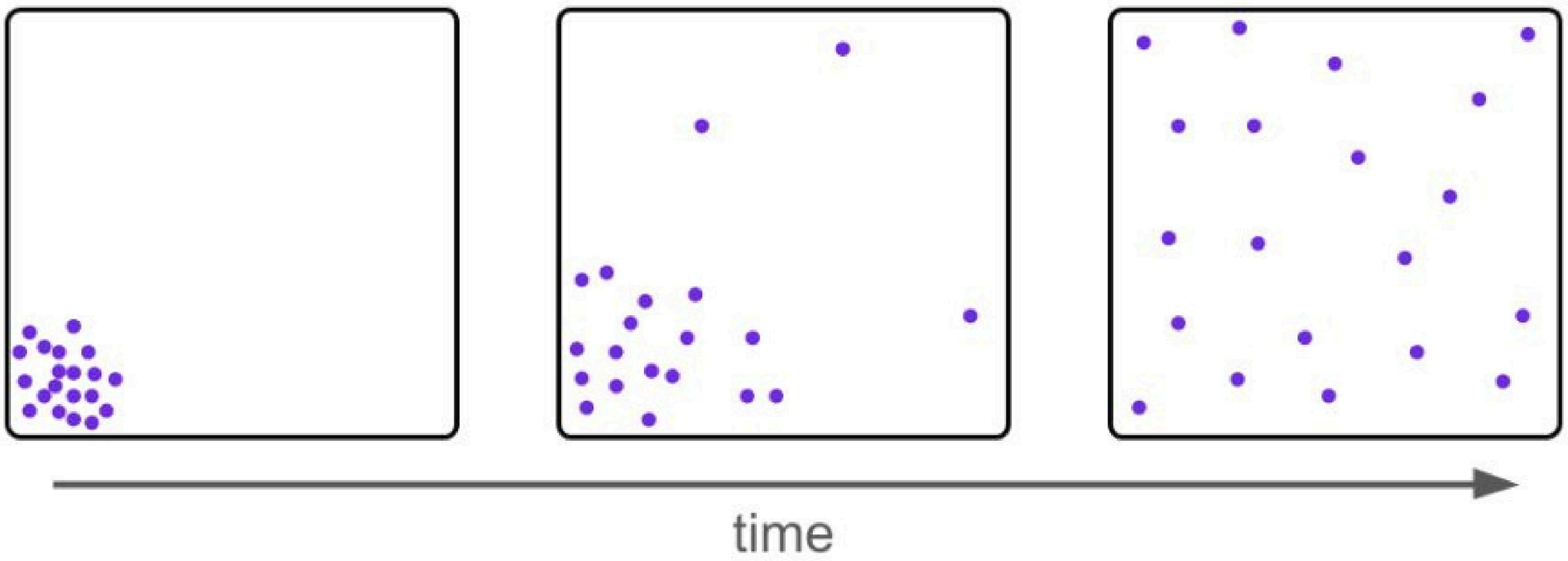
[OpenAI’s DALLE-2]

How Diffusion Models work?

Diffusion Models draw inspiration from **thermodynamics/statistical physics**, e.g. any localized distribution of a particles will eventually spread out to fill an entire space evenly simply through random motion.

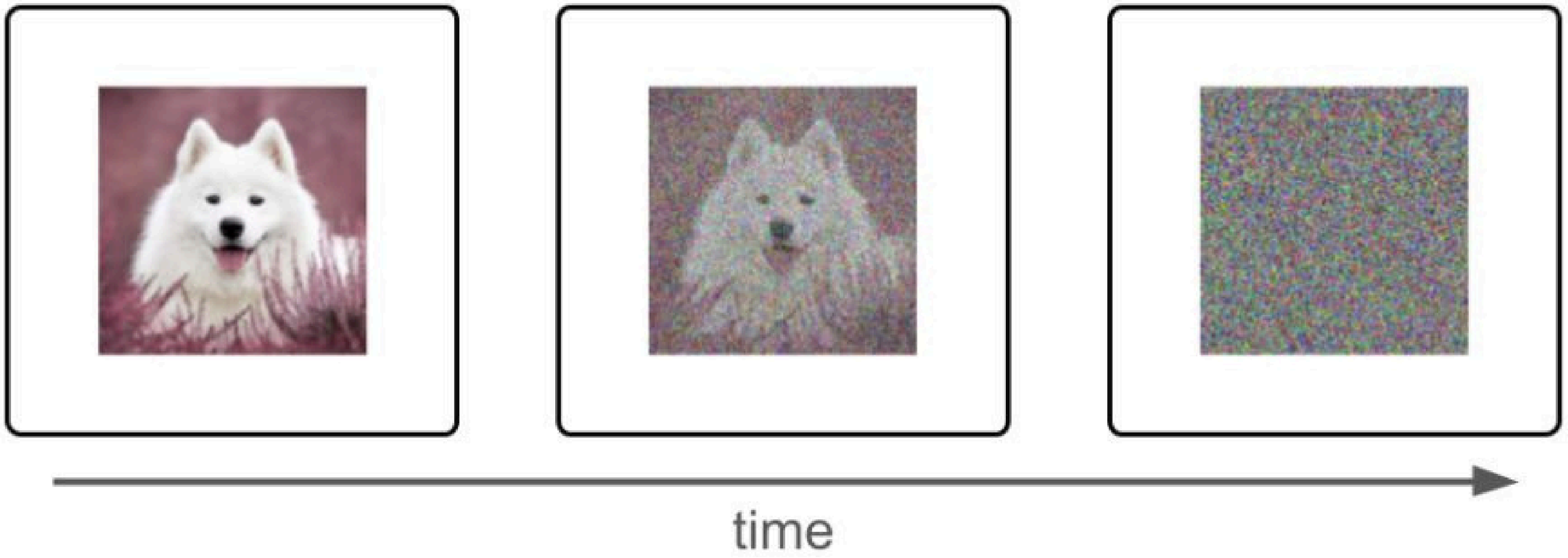


Diffusion Models draw inspiration from **thermodynamics/statistical physics**, e.g. any localized distribution of a particles will eventually spread out to fill an entire space evenly simply through random motion.



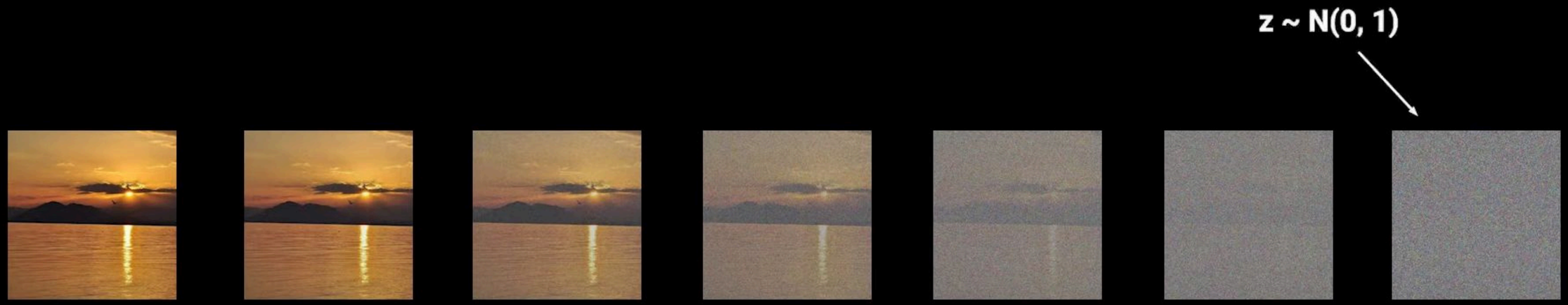
Can it be reversed? **In Physics, No. In Machine Learning, Yes.**

Diffusion Models place images in a **"high dimensional room"**, corrupting them with random noise in a similar way.



The goal is to train a Machine Learning model that learns how to go **“backwards through time”**, reversing this corruption process.

Forward Process (Noising) →



←Reverse Process (Denoising)

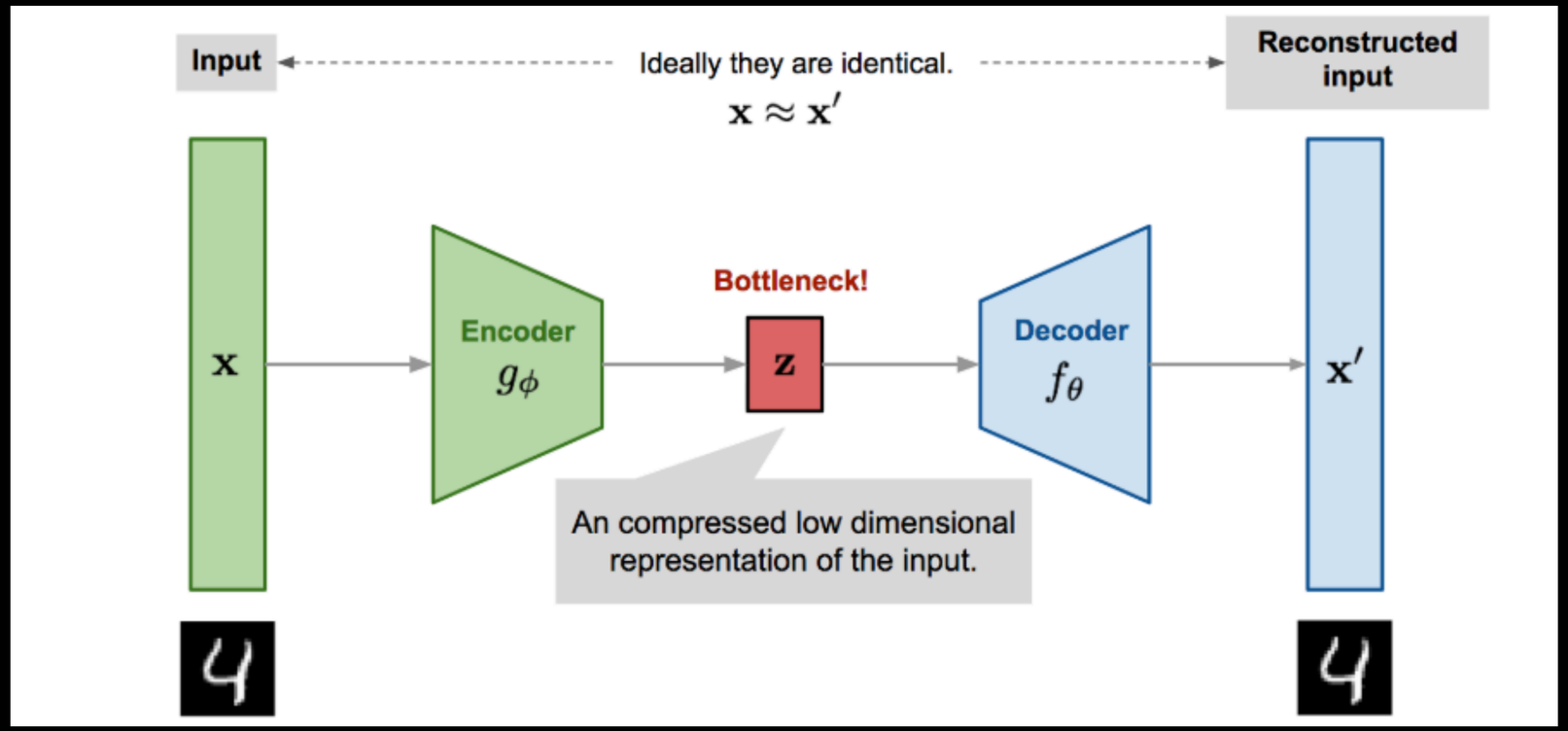
Markov Chain

Auto-Encoder

Autoencoder is a neural network designed to learn an identity function in an unsupervised way to reconstruct the original input while compressing the data in the process so as to discover a more efficient and compressed representation

It consists of two networks:

- Encoder network: It translates the original high-dimension input into the latent low-dimensional code. The input size is larger than the output size.
- Decoder network: The decoder network recovers the data from the code, likely with larger and larger output layers.



Variational Auto-Encoder

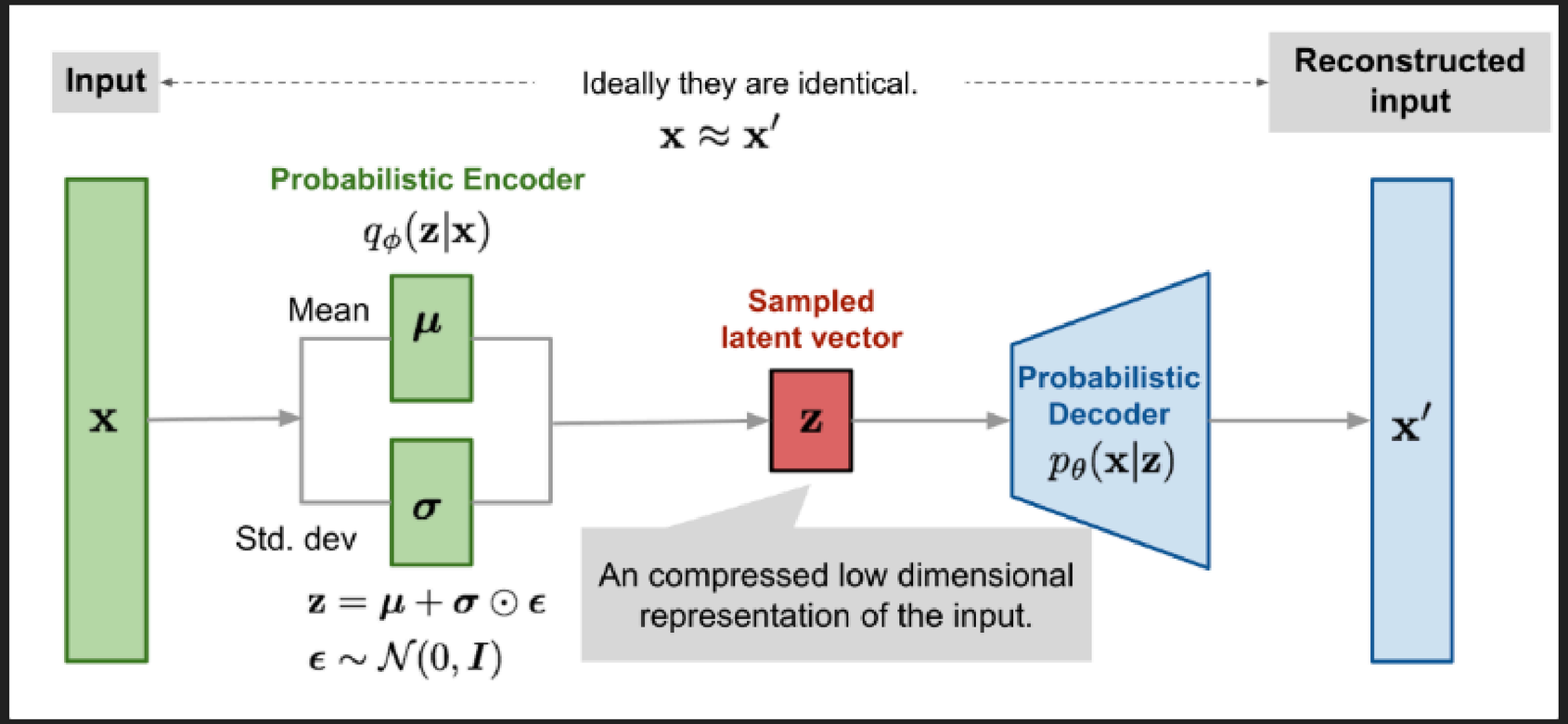
he idea of Variational Autoencoder (Kingma & Welling, 2014), short for VAE, is actually less similar to all the autoencoder models above, but deeply rooted in the methods of variational bayesian and graphical model.

Instead of mapping the input into a fixed vector, we want to map it into a distribution. Let's label this distribution as p_θ, parameterized by θ. The relationship between the data input x and the latent encoding vector z can be fully defined by:

Prior $p_\theta(z)$

Likelihood $p_\theta(x \mid z)$

Posterior $p_\theta(z \mid x)$



Assuming that we know the real parameter θ^* for this distribution. In order to generate a sample that looks like a real data point $\mathbf{x}^{(i)}$, we follow these steps:

1. First, sample a $\mathbf{z}^{(i)}$ from a prior distribution $p_{\theta^*}(\mathbf{z})$.
2. Then a value $\mathbf{x}^{(i)}$ is generated from a conditional distribution $p_{\theta^*}(\mathbf{x}|\mathbf{z} = \mathbf{z}^{(i)})$.

The optimal parameter θ^* is the one that maximizes the probability of generating real data samples:

$$\theta^* = \arg\max_{\theta} \prod_{i=1}^{n} p_\theta(\mathbf{x}^{(i)})$$

Commonly we use the log probabilities to convert the product on RHS to a sum:

$$\theta^* = \arg\max_{\theta} \sum_{i=1}^{n} \log p_\theta(\mathbf{x}^{(i)})$$

Now let's update the equation to better demonstrate the data generation process so as to involve the encoding vector:

$$p_\theta(\mathbf{x}^{(i)}) = \int p_\theta(\mathbf{x}^{(i)}|\mathbf{z}) p_\theta(\mathbf{z}) d\mathbf{z}$$

$$\log p(x) = log \int p(x, z) dz$$

$$\log p(x) = log \int \frac{q(z \mid x)}{q(z \mid x)} p(x, z) dz$$

$$\log p(x) = log \mathbb{E}_{q(z|x)} \left[\frac{p(x, z)}{q(z \mid x)} \right]$$

Jensen's Inequality

$$\mathbb{E}[f(x)] \geq f(\mathbb{E}[x])$$

Evidence Lower Bound
(ELBO)

Evidence

$$\log p(x) \geq \mathbb{E}_{q(z|x)} \left[\log \frac{p(x,z)}{q(z \mid x)} \right]$$

$$\mathcal{L}(x)$$

$$\mathcal{L}(x) = \mathbb{E}_{q(z|x)} \left[\log \frac{p(x \mid z).p(z)}{q(z \mid x)} \right]$$

$$\mathcal{L}(x) = \mathbb{E}_{q(z|x)} \Big[\log p(x \mid z) \Big] - \mathrm{KL}(q(z \mid x) \mid p(z))$$

Reconstruction
Loss

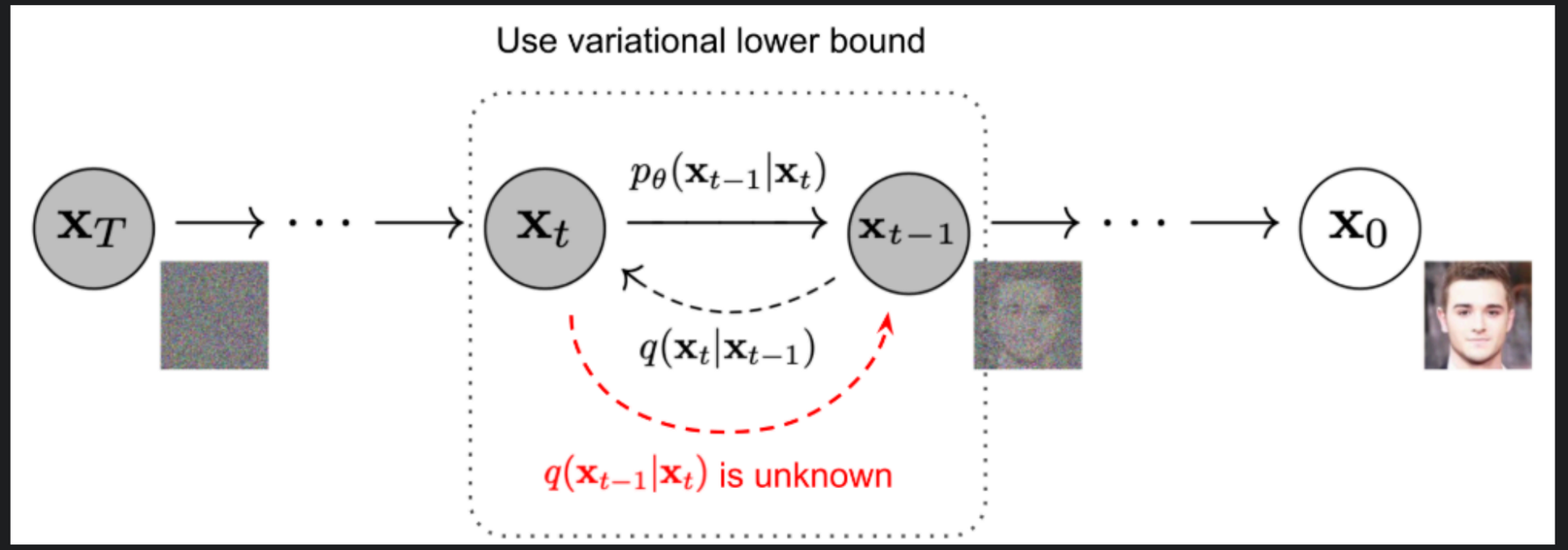
KL Divergence
Loss

Forward diffusion process

Given a data point sampled from a real data distribution $\mathbf{x}_0 \sim q(\mathbf{x})$, let us define a *forward diffusion process* in which we add small amount of Gaussian noise to the sample in T steps, producing a sequence of noisy samples $\mathbf{x}_1, \dots, \mathbf{x}_T$. The step sizes are controlled by a variance schedule $\{\beta_t \in (0,1)\}_{t=1}^T$.

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}) \quad q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1})$$

The data sample $\mathbf{x}_0$ gradually loses its distinguishable features as the step t becomes larger. Eventually when $T \to \infty$, $\mathbf{x}_T$ is equivalent to an isotropic Gaussian distribution.



A nice property of the above process is that we can sample $\mathbf{x}_t$ at any arbitrary time step t in a closed form using reparameterization trick. Let $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$:

$$
\begin{aligned}
\mathbf{x}_t &= \sqrt{\alpha_t}\mathbf{x}_{t-1} + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}_{t-1} && ;\text{where } \boldsymbol{\epsilon}_{t-1}, \boldsymbol{\epsilon}_{t-2}, \cdots \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \\
&= \sqrt{\alpha_t\alpha_{t-1}}\mathbf{x}_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\bar{\boldsymbol{\epsilon}}_{t-2} && ;\text{where } \bar{\boldsymbol{\epsilon}}_{t-2} \text{ merges two Gaussians } (*). \\
&= \dots \\
&= \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon} \\
q(\mathbf{x}_t|\mathbf{x}_0) &= \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I})
\end{aligned}
$$

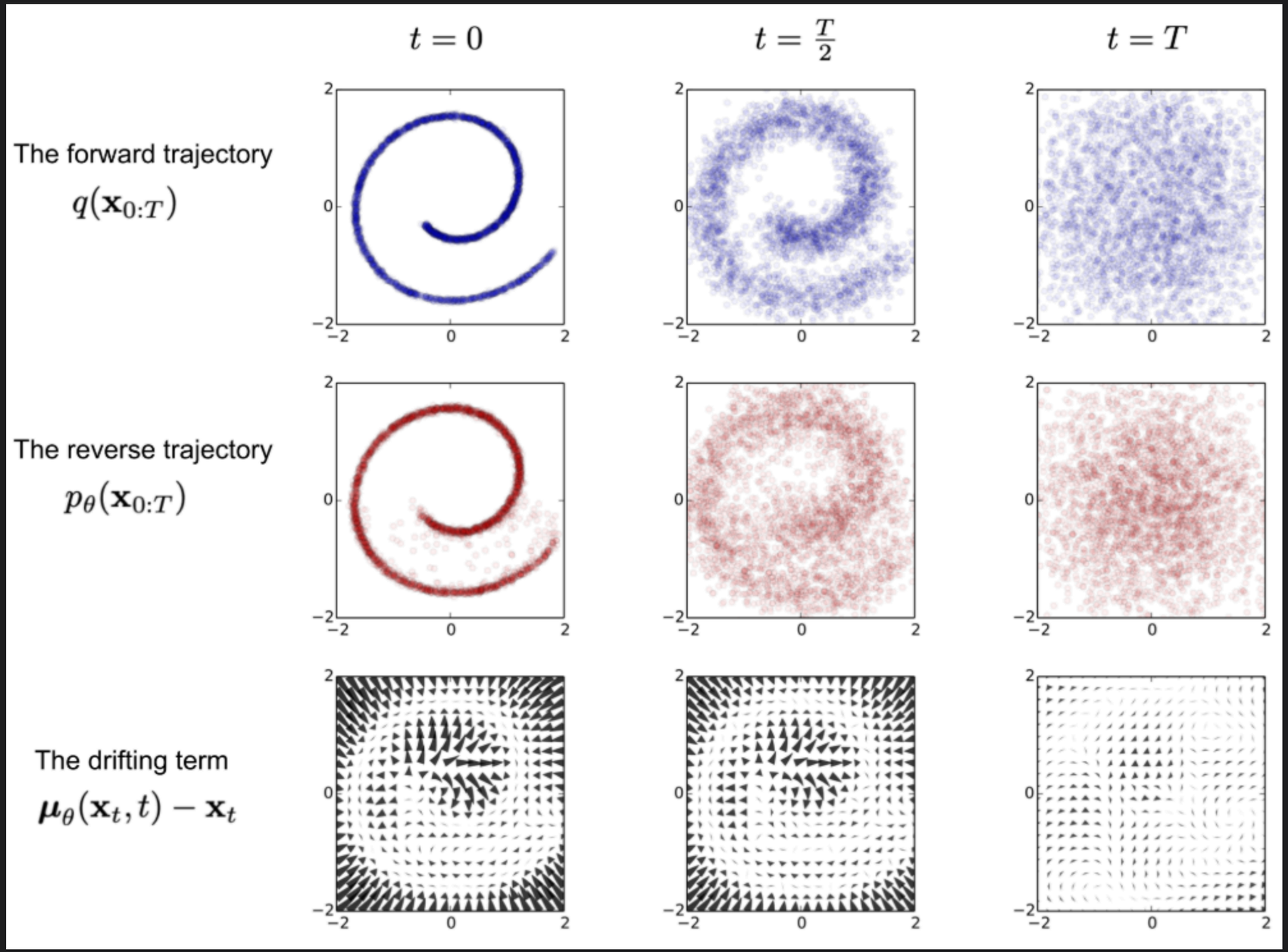
(*) Recall that when we merge two Gaussians with different variance, $\mathcal{N}(\mathbf{0}, \sigma_1^2\mathbf{I})$ and $\mathcal{N}(\mathbf{0}, \sigma_2^2\mathbf{I})$, the new distribution is $\mathcal{N}(\mathbf{0}, (\sigma_1^2 + \sigma_2^2)\mathbf{I})$. Here the merged standard deviation is $\sqrt{(1-\alpha_t) + \alpha_t(1-\alpha_{t-1})} = \sqrt{1-\alpha_t\alpha_{t-1}}$.

Usually, we can afford a larger update step when the sample gets noisier, so $\beta_1 < \beta_2 < \cdots < \beta_T$ and therefore $\bar{\alpha}_1 > \cdots > \bar{\alpha}_T$.

Reverse diffusion process

If we can reverse the above process and sample from $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$, we will be able to recreate the true sample from a Gaussian noise input, $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. Note that if β_t is small enough, $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ will also be Gaussian. Unfortunately, we cannot easily estimate $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ because it needs to use the entire dataset and therefore we need to learn a model p_θ to approximate these conditional probabilities in order to run the *reverse diffusion process*.

$$p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) \quad p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t))$$


$t = 0$
$t = \frac{T}{2}$
$t = T$
The forward trajectory
$q(\mathbf{x}_{0:T})$
The reverse trajectory
$p_\theta(\mathbf{x}_{0:T})$
The drifting term
$\boldsymbol{\mu}_\theta(\mathbf{x}_t, t) - \mathbf{x}_t$

It is noteworthy that the reverse conditional probability is tractable when conditioned on $\mathbf{x}_0$:

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1}; \tilde{\boldsymbol{\mu}}(\mathbf{x}_t, \mathbf{x}_0), \tilde{\beta}_t \mathbf{I})$$

Using Bayes' rule, we have:

$$\begin{aligned}
q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) &= q(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_0) \frac{q(\mathbf{x}_{t-1}|\mathbf{x}_0)}{q(\mathbf{x}_t|\mathbf{x}_0)} \\
&\propto \exp\Big(-\frac{1}{2}\big(\frac{(\mathbf{x}_t - \sqrt{\alpha_t}\mathbf{x}_{t-1})^2}{\beta_t} + \frac{(\mathbf{x}_{t-1} - \sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0)^2}{1-\bar{\alpha}_{t-1}} - \frac{(\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0)^2}{1-\bar{\alpha}_t}\big)\Big) \\
&= \exp\Big(-\frac{1}{2}\big(\frac{\mathbf{x}_t^2 - 2\sqrt{\alpha_t}\mathbf{x}_t\mathbf{x}_{t-1} + \alpha_t \mathbf{x}_{t-1}^2}{\beta_t} + \frac{\mathbf{x}_{t-1}^2 - 2\sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0\mathbf{x}_{t-1} + \bar{\alpha}_{t-1}\mathbf{x}_0^2}{1-\bar{\alpha}_{t-1}} - \frac{(\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0)^2}{1-\bar{\alpha}_t}\big)\Big) \\
&= \exp\Big(-\frac{1}{2}\big((\frac{\alpha_t}{\beta_t} + \frac{1}{1-\bar{\alpha}_{t-1}})\mathbf{x}_{t-1}^2 - (\frac{2\sqrt{\alpha_t}}{\beta_t}\mathbf{x}_t + \frac{2\sqrt{\bar{\alpha}_{t-1}}}{1-\bar{\alpha}_{t-1}}\mathbf{x}_0)\mathbf{x}_{t-1} + C(\mathbf{x}_t, \mathbf{x}_0)\big)\Big)
\end{aligned}$$

where $C(\mathbf{x}_t, \mathbf{x}_0)$ is some function not involving $\mathbf{x}_{t-1}$ and details are omitted. Following the standard Gaussian density function, the mean and variance can be parameterized as follows (recall that $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$):

$$
\begin{aligned}
\tilde{\beta}_t &= 1/(\frac{\alpha_t}{\beta_t} + \frac{1}{1 - \bar{\alpha}_{t-1}}) = 1/(\frac{\alpha_t - \bar{\alpha}_t + \beta_t}{\beta_t(1 - \bar{\alpha}_{t-1})}) = \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t} \cdot \beta_t \\
\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) &= (\frac{\sqrt{\alpha_t}}{\beta_t}\mathbf{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}}{1 - \bar{\alpha}_{t-1}}\mathbf{x}_0)/(\frac{\alpha_t}{\beta_t} + \frac{1}{1 - \bar{\alpha}_{t-1}}) \\
&= (\frac{\sqrt{\alpha_t}}{\beta_t}\mathbf{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}}{1 - \bar{\alpha}_{t-1}}\mathbf{x}_0)\frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t} \cdot \beta_t \\
&= \frac{\sqrt{\alpha_t}(1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t}\mathbf{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1 - \bar{\alpha}_t}\mathbf{x}_0
\end{aligned}
$$

Thanks to the nice property, we can represent $\mathbf{x}_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}(\mathbf{x}_t - \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\epsilon}_t)$ and plug it into the above equation and obtain:

$$
\begin{aligned}
\tilde{\boldsymbol{\mu}}_t &= \frac{\sqrt{\alpha_t}(1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t}\mathbf{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1 - \bar{\alpha}_t}\frac{1}{\sqrt{\bar{\alpha}_t}}(\mathbf{x}_t - \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\epsilon}_t) \\
&= \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}}\boldsymbol{\epsilon}_t\right)
\end{aligned}
$$

From the forward diffusion image, such a setup is very similar to VAE and thus we can use the variational lower bound to optimize the negative log-likelihood.

$$
\begin{aligned}
-\log p_\theta(\mathbf{x}_0) &\leq -\log p_\theta(\mathbf{x}_0) + D_{\mathrm{KL}}(q(\mathbf{x}_{1:T}|\mathbf{x}_0)\|p_\theta(\mathbf{x}_{1:T}|\mathbf{x}_0)) && \text{; KL is non-negative} \\
&= -\log p_\theta(\mathbf{x}_0) + \mathbb{E}_{\mathbf{x}_{1:T}\sim q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})/p_\theta(\mathbf{x}_0)}\right] \\
&= -\log p_\theta(\mathbf{x}_0) + \mathbb{E}_q\left[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})} + \log p_\theta(\mathbf{x}_0)\right] \\
&= \mathbb{E}_q\left[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})}\right] \\
\text{Let } L_{\mathrm{VLB}} &= \mathbb{E}_{q(\mathbf{x}_{0:T})}\left[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})}\right] \geq -\mathbb{E}_{q(\mathbf{x}_0)} \log p_\theta(\mathbf{x}_0)
\end{aligned}
$$

It is also straightforward to get the same result using Jensen's inequality. Say we want to minimize the cross entropy as the learning objective,

$$
\begin{aligned}
L_{\text{CE}} &= -\mathbb{E}_{q(\mathbf{x}_0)} \log p_\theta(\mathbf{x}_0) \\
&= -\mathbb{E}_{q(\mathbf{x}_0)} \log \Big(\int p_\theta(\mathbf{x}_{0:T}) d\mathbf{x}_{1:T} \Big) \\
&= -\mathbb{E}_{q(\mathbf{x}_0)} \log \Big(\int q(\mathbf{x}_{1:T}\vert\mathbf{x}_0) \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}\vert\mathbf{x}_0)} d\mathbf{x}_{1:T} \Big) \\
&= -\mathbb{E}_{q(\mathbf{x}_0)} \log \Big(\mathbb{E}_{q(\mathbf{x}_{1:T}\vert\mathbf{x}_0)} \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}\vert\mathbf{x}_0)} \Big) \\
&\leq -\mathbb{E}_{q(\mathbf{x}_{0:T})} \log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}\vert\mathbf{x}_0)} \\
&= \mathbb{E}_{q(\mathbf{x}_{0:T})} \Big[\log \frac{q(\mathbf{x}_{1:T}\vert\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})} \Big] = L_{\text{VLB}}
\end{aligned}
$$

To convert each term in the equation to be analytically computable, the objective can be further rewritten to be a combination of several KL-divergence and entropy terms (See the detailed step-by-step process in Appendix B in Sohl-Dickstein et al., 2015):

$$
\begin{aligned}
L_{\text{VLB}} &= \mathbb{E}_{q(\mathbf{x}_{0:T})}\Big[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})}\Big] \\
&= \mathbb{E}_q\Big[\log \frac{\prod_{t=1}^T q(\mathbf{x}_t|\mathbf{x}_{t-1})}{p_\theta(\mathbf{x}_T)\prod_{t=1}^T p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}\Big] \\
&= \mathbb{E}_q\Big[-\log p_\theta(\mathbf{x}_T) + \sum_{t=1}^T \log \frac{q(\mathbf{x}_t|\mathbf{x}_{t-1})}{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}\Big] \\
&= \mathbb{E}_q\Big[-\log p_\theta(\mathbf{x}_T) + \sum_{t=2}^T \log \frac{q(\mathbf{x}_t|\mathbf{x}_{t-1})}{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)} + \log \frac{q(\mathbf{x}_1|\mathbf{x}_0)}{p_\theta(\mathbf{x}_0|\mathbf{x}_1)}\Big] \\
&= \mathbb{E}_q\Big[-\log p_\theta(\mathbf{x}_T) + \sum_{t=2}^T \log \Big(\frac{q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)}{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)} \cdot \frac{q(\mathbf{x}_t|\mathbf{x}_0)}{q(\mathbf{x}_{t-1}|\mathbf{x}_0)}\Big) + \log \frac{q(\mathbf{x}_1|\mathbf{x}_0)}{p_\theta(\mathbf{x}_0|\mathbf{x}_1)}\Big] \\
&= \mathbb{E}_q\Big[-\log p_\theta(\mathbf{x}_T) + \sum_{t=2}^T \log \frac{q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)}{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)} + \sum_{t=2}^T \log \frac{q(\mathbf{x}_t|\mathbf{x}_0)}{q(\mathbf{x}_{t-1}|\mathbf{x}_0)} + \log \frac{q(\mathbf{x}_1|\mathbf{x}_0)}{p_\theta(\mathbf{x}_0|\mathbf{x}_1)}\Big] \\
&= \mathbb{E}_q\Big[-\log p_\theta(\mathbf{x}_T) + \sum_{t=2}^T \log \frac{q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)}{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)} + \log \frac{q(\mathbf{x}_T|\mathbf{x}_0)}{q(\mathbf{x}_1|\mathbf{x}_0)} + \log \frac{q(\mathbf{x}_1|\mathbf{x}_0)}{p_\theta(\mathbf{x}_0|\mathbf{x}_1)}\Big] \\
&= \mathbb{E}_q\Big[\log \frac{q(\mathbf{x}_T|\mathbf{x}_0)}{p_\theta(\mathbf{x}_T)} + \sum_{t=2}^T \log \frac{q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)}{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)} - \log p_\theta(\mathbf{x}_0|\mathbf{x}_1)\Big] \\
&= \mathbb{E}_q[\underbrace{D_{\text{KL}}(q(\mathbf{x}_T|\mathbf{x}_0) \parallel p_\theta(\mathbf{x}_T))}_{L_T} + \sum_{t=2}^T \underbrace{D_{\text{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) \parallel p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))}_{L_{t-1}} \underbrace{- \log p_\theta(\mathbf{x}_0|\mathbf{x}_1)}_{L_0}]
\end{aligned}
$$

Let's label each component in the variational lower bound loss separately:

$$
\begin{aligned}
L_{\text{VLB}} &= L_T + L_{T-1} + \cdots + L_0 \\
\text{where } L_T &= D_{\text{KL}}(q(\mathbf{x}_T|\mathbf{x}_0) \parallel p_\theta(\mathbf{x}_T)) \\
L_t &= D_{\text{KL}}(q(\mathbf{x}_t|\mathbf{x}_{t+1}, \mathbf{x}_0) \parallel p_\theta(\mathbf{x}_t|\mathbf{x}_{t+1})) \text{ for } 1 \leq t \leq T-1 \\
L_0 &= -\log p_\theta(\mathbf{x}_0|\mathbf{x}_1)
\end{aligned}
$$

Every KL term in L_{VLB} (except for L_0) compares two Gaussian distributions and therefore they can be computed in closed form. L_T is constant and can be ignored during training because q has no learnable parameters and $\mathbf{x}_T$ is a Gaussian noise. Ho et al. 2020 models L_0 using a separate discrete decoder derived from $\mathcal{N}(\mathbf{x}_0; \boldsymbol{\mu}_\theta(\mathbf{x}_1, 1), \boldsymbol{\Sigma}_\theta(\mathbf{x}_1, 1))$.

Parameterization of L_t for Training Loss

Recall that we need to learn a neural network to approximate the conditioned probability distributions in the reverse diffusion process, $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t))$. We would like to train $\boldsymbol{\mu}_\theta$ to predict $\tilde{\boldsymbol{\mu}}_t = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_t\right)$. Because $\mathbf{x}_t$ is available as input at training time, we can reparameterize the Gaussian noise term instead to make it predict $\boldsymbol{\epsilon}_t$ from the input $\mathbf{x}_t$ at time step t:

$$\boldsymbol{\mu}_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right)$$
$$\text{Thus } \mathbf{x}_{t-1} = \mathcal{N}(\mathbf{x}_{t-1}; \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t))$$

The loss term L_t is parameterized to minimize the difference from $\tilde{\boldsymbol{\mu}}$:

$$\begin{aligned}
L_t &= \mathbb{E}_{\mathbf{x}_0,\boldsymbol{\epsilon}}\left[\frac{1}{2\|\boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t)\|_2^2}\|\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) - \boldsymbol{\mu}_\theta(\mathbf{x}_t, t)\|^2\right] \\
&= \mathbb{E}_{\mathbf{x}_0,\boldsymbol{\epsilon}}\left[\frac{1}{2\|\boldsymbol{\Sigma}_\theta\|_2^2}\|\frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_t\right) - \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right)\|^2\right] \\
&= \mathbb{E}_{\mathbf{x}_0,\boldsymbol{\epsilon}}\left[\frac{(1-\alpha_t)^2}{2\alpha_t(1-\bar{\alpha}_t)\|\boldsymbol{\Sigma}_\theta\|_2^2}\|\boldsymbol{\epsilon}_t - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\|^2\right] \\
&= \mathbb{E}_{\mathbf{x}_0,\boldsymbol{\epsilon}}\left[\frac{(1-\alpha_t)^2}{2\alpha_t(1-\bar{\alpha}_t)\|\boldsymbol{\Sigma}_\theta\|_2^2}\|\boldsymbol{\epsilon}_t - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_t, t)\|^2\right]
\end{aligned}$$

Simplification

Empirically, Ho et al. (2020) found that training the diffusion model works better with a simplified objective that ignores the weighting term:

$$
\begin{aligned}
L_t^{\text{simple}} &= \mathbb{E}_{t\sim[1,T],\mathbf{x}_0,\boldsymbol{\epsilon}_t}\left[\|\boldsymbol{\epsilon}_t - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\|^2\right] \\
&= \mathbb{E}_{t\sim[1,T],\mathbf{x}_0,\boldsymbol{\epsilon}_t}\left[\|\boldsymbol{\epsilon}_t - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_t, t)\|^2\right]
\end{aligned}
$$

The final simple objective is:

$$
L_{\text{simple}} = L_t^{\text{simple}} + C
$$

where C is a constant not depending on θ.

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
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** $\mathbf{x}_0$

Classifier Guidance in Diffusion

Feature	Classifier-Guided	Classifier-Free Guidance
Need to train another model?	Yes, a classifier needs to be trained using noisy images.	Not really, for example, CLIP can be used directly for text-to-image tasks.
Need to retrain the diffusion model?	No, pre-trained diffusion models are usable as is.	Yes, diffusion needs to be retrained using this method.
Control over final output	Can control the generated category. The number of classes the classifier can identify is the number of classes you can control in generation.	Any (almost) condition can be controlled.

Classifier Based Guidance

For the ease of introducing classifier guidance and classifier-free guidance, we choose to estimate the mean of the Gaussian distribution via score-based estimation:

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}} x_t + \frac{1-\alpha}{\sqrt{\alpha_t}} \nabla \log p(x)$$

where the gradient of log distribution of x_t is called ‘score’ and usually not directly available, but can be approximated by learning a score function via neural networks:

$$\nabla \log p(x) \approx \boldsymbol{s}_\theta(x_t, t)$$

and then the mean of the Gaussian distribution becomes:

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}} x_t + \frac{1-\alpha}{\sqrt{\alpha_t}} s_\theta(x_t, t)$$

After introducing the condition y, the probability p(x_t) in the gradient becomes conditional probability p(x_t | y):

$$\nabla_{x_t} \log p(x_t) \rightarrow \nabla_{x_t} \log p(x_t \mid y)$$

By apply Bayes' Theorem, we obtain a new score function:

$$\nabla \log p(x_t \mid y) = \nabla \log \left(\frac{p(x_t) p(y \mid x_t)}{p(y)} \right) = \nabla \log p(x_t) + \nabla \log p(y \mid x_t)$$

To control the strength of guidance, we can add a guidance_scale parameter:

$$\nabla \log p(x_t) + \lambda \nabla \log p(y \mid x_t)$$

Classifier Free Guidance

Classifier-free guidance only requires a slight modification on the basis of classifier guidance. By manipulating the gradient, we can obtain the equation below

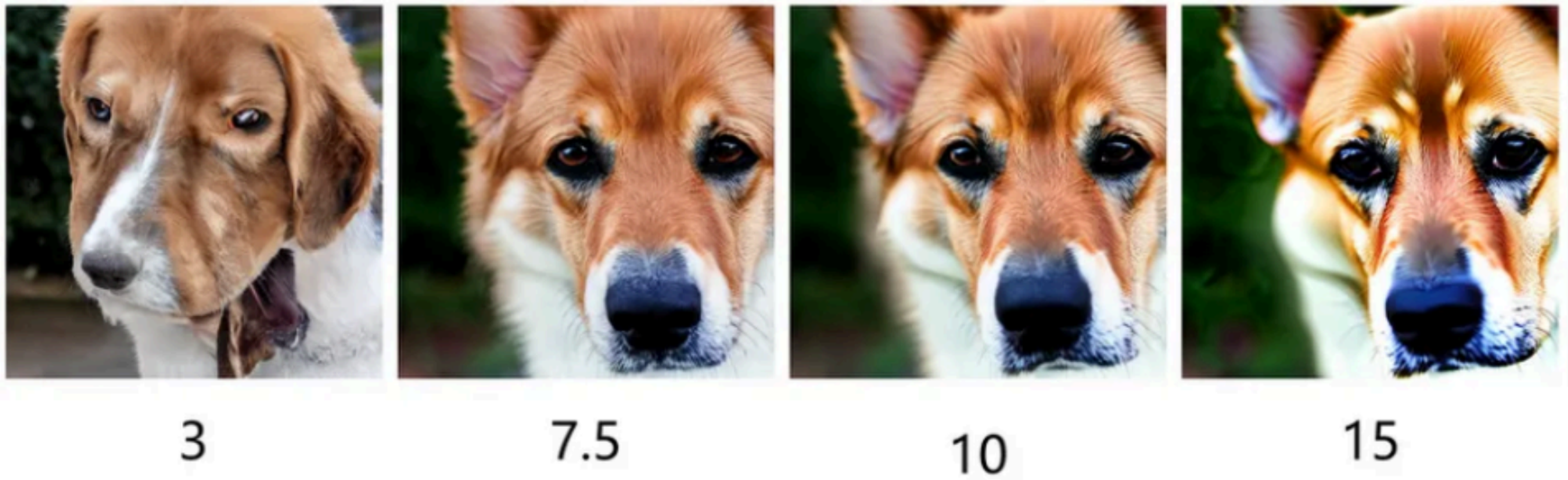
$$\nabla_{x_t} \log p(y \mid x_t) = \nabla_{x_t} \log p(x_t \mid y) - \nabla_{x_t} \log p(x_t)$$

$$\begin{aligned}
\nabla_{x_t} \log p(x_t|y) &= \nabla_{x_t} \log p(x_t) + \gamma \left(\nabla_{x_t} \log p(x_t|y) - \nabla \log p(x_t) \right) \\
&= \nabla_{x_t} \log p(x_t) + \gamma \nabla \log p(x_t|y) - \gamma \nabla_{x_t} \log p(x_t) \\
&= \gamma \nabla_{x_t} \log p(x_t|y) - (\gamma - 1) \nabla_{x_t} \log p(x_t) \\
&\approx \gamma \underbrace{s_\theta(s_t, y, t)}_{\text{conditional score}} - (\gamma - 1) \underbrace{s_\theta(s_t, \emptyset, t)}_{\text{unconditional score}} \\
&= \underbrace{s_\theta(s_t, \emptyset, t)}_{\text{unconditional score}} + \gamma \Big(\underbrace{s_\theta(s_t, y, t)}_{\text{conditional score}} - \underbrace{s_\theta(s_t, \emptyset, t)}_{\text{unconditional score}} \Big)
\end{aligned}$$

Classifier Guidance can only control the categories generated by the classification model. If the classification model distinguishes 10 classes, then Classifier Guidance can only guide the diffusion model to generate those fixed 10 classes; it won't work for an additional class.

Classifier-Free Guidance is a powerful method. Although it requires retraining the diffusion model, once trained, it can take off directly without any limitation on the number of categories. It can work even when the condition is text or image.

Classifier-Free Guidance is a powerful method. Although it requires retraining the diffusion model, once trained, it can take off directly without any limitation on the number of categories. It can work even when the condition is text or image.



The image generated from Stable Diffusion v1.5 under different guidance scale. Image source: https://zhuanlan.zhihu.com/p/660518657

References

- From Autoencoder to Beta-VAE: https://lilianweng.github.io/posts/2018-08-12-vae/
- What are Diffusion Models?: https://lilianweng.github.io/posts/2021-07-11-diffusion-models/
- DDPM Paper: https://hojonathanho.github.io/diffusion/
- High-Resolution Image Synthesis with Latent Diffusion Models: https://arxiv.org/pdf/2112.10752
- Explaining AI Diffusion Playlist: https://www.youtube.com/playlist?list=PL8VDJoEXIjpo2S7X-1YKZnbHyLGyESDCe
- Stable Diffusion Model Codebase: https://github.com/explainingai-code/StableDiffusion-PyTorch