VAE AND DIFFUSION MODELS

Sergey Nikolenko

Harbour Space University

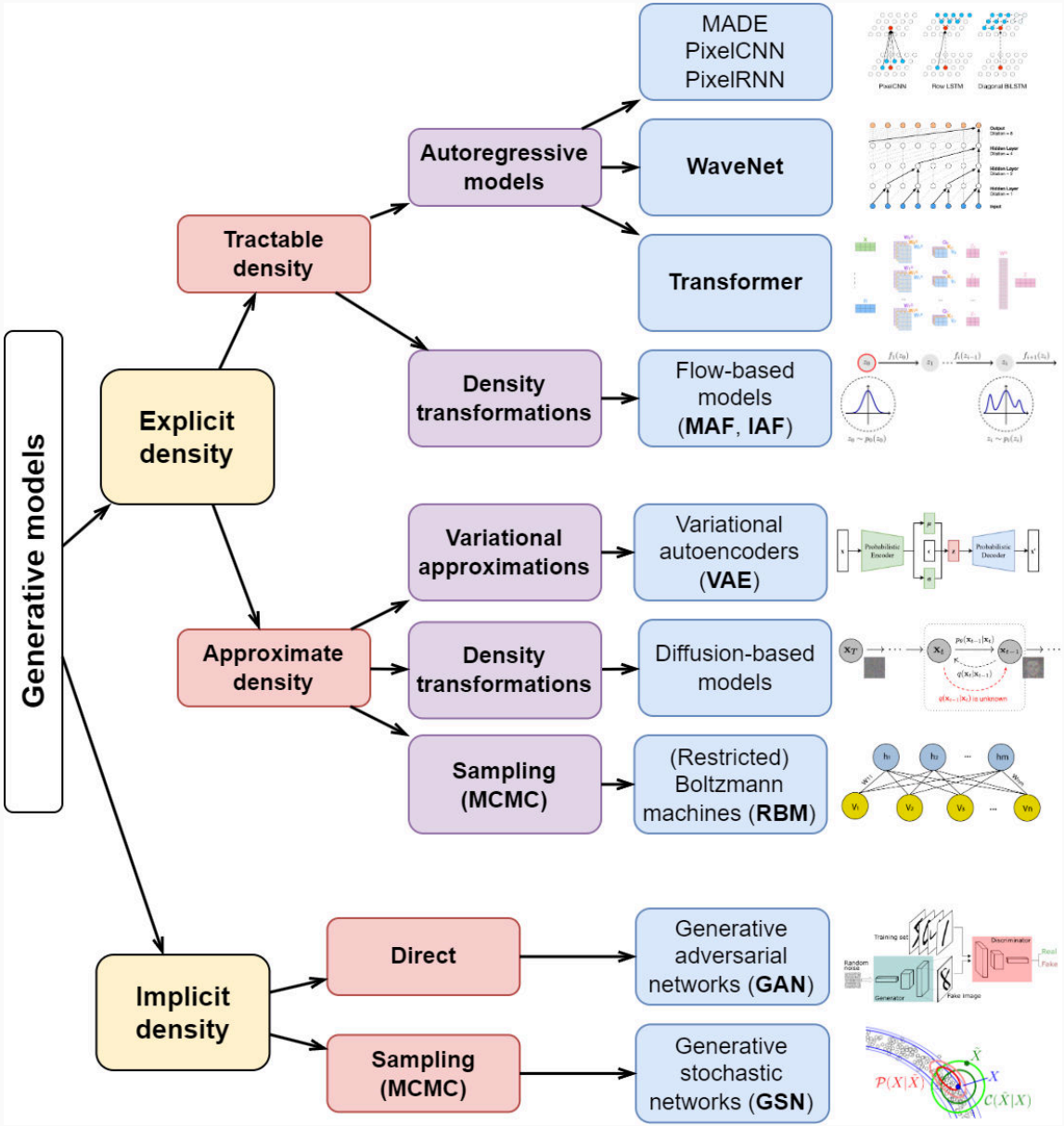
June 24, 2026

Random facts:

- on June 24, 1374, a "dancing plague" broke out in Aachen: crowds of people danced uncontrollably through the streets for days, some until they collapsed, in one of several genuinely documented outbreaks of compulsive dancing in medieval Europe
- on June 24, 1497, John Cabot, sailing under an English flag, reached the coast of Newfoundland and became the first European on the North American mainland since the Vikings; he was rewarded by King Henry VII with the grand sum of ten pounds
- on June 24, 1717, the first Masonic Grand Lodge in the world, the Premier Grand Lodge of England, was founded in London
- on June 24, 1812, Napoleon led his Grande Armée of around 600,000 men across the Niemen River into Russia; less than a tenth of them would return
- on June 24, 1947, the pilot Kenneth Arnold reported nine objects flying "like a saucer skipping over water" near Mount Rainier; a newspaper turned his phrasing into "flying saucer" and the modern UFO era was born
- on June 24, 2010, at Wimbledon, John Isner of the United States defeated Nicolas Mahut of France in the longest match in professional tennis history, 70:68 in the final fifth set

Generative models in deep learning

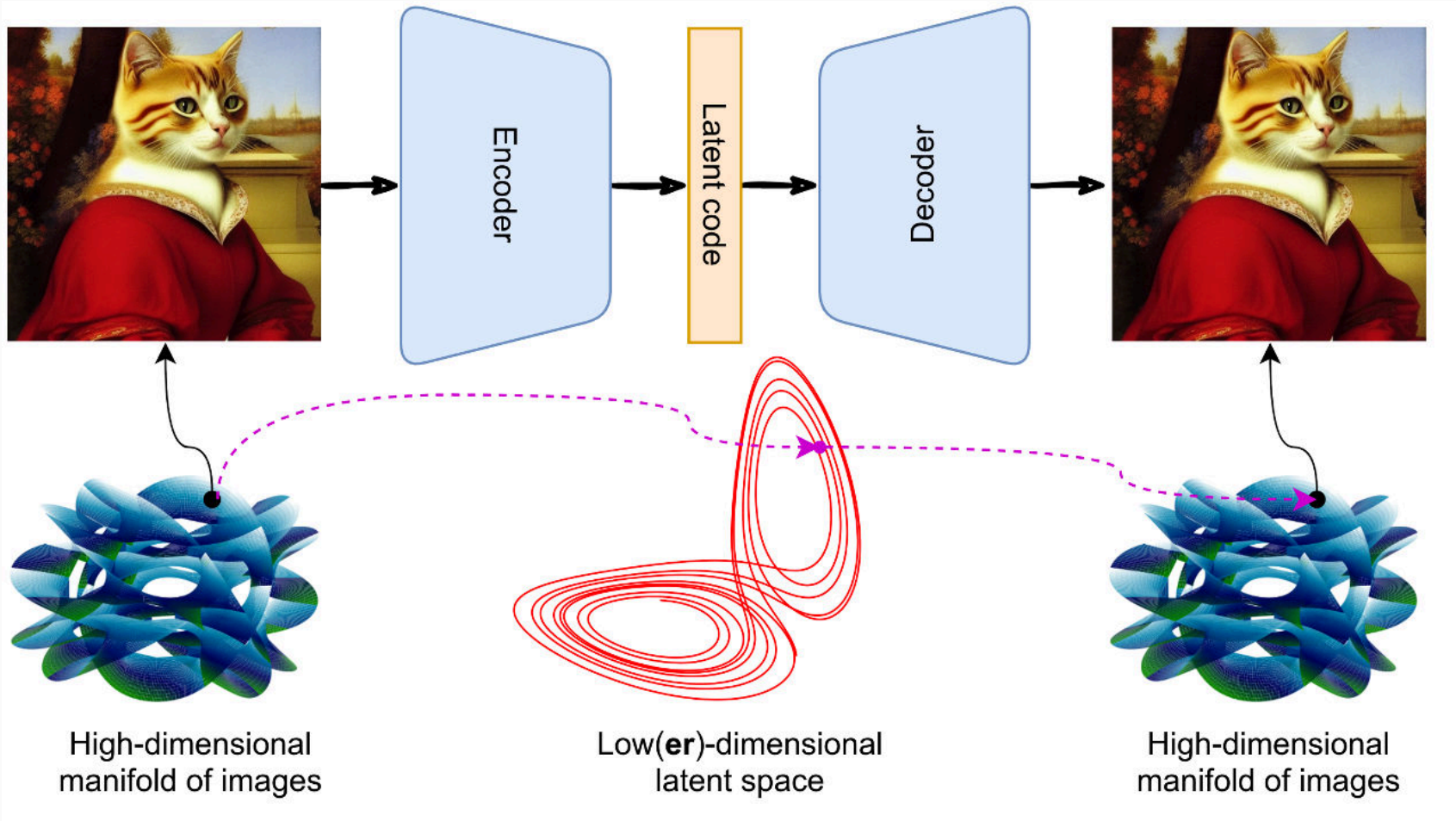
Deep generative models



Variational autoencoders

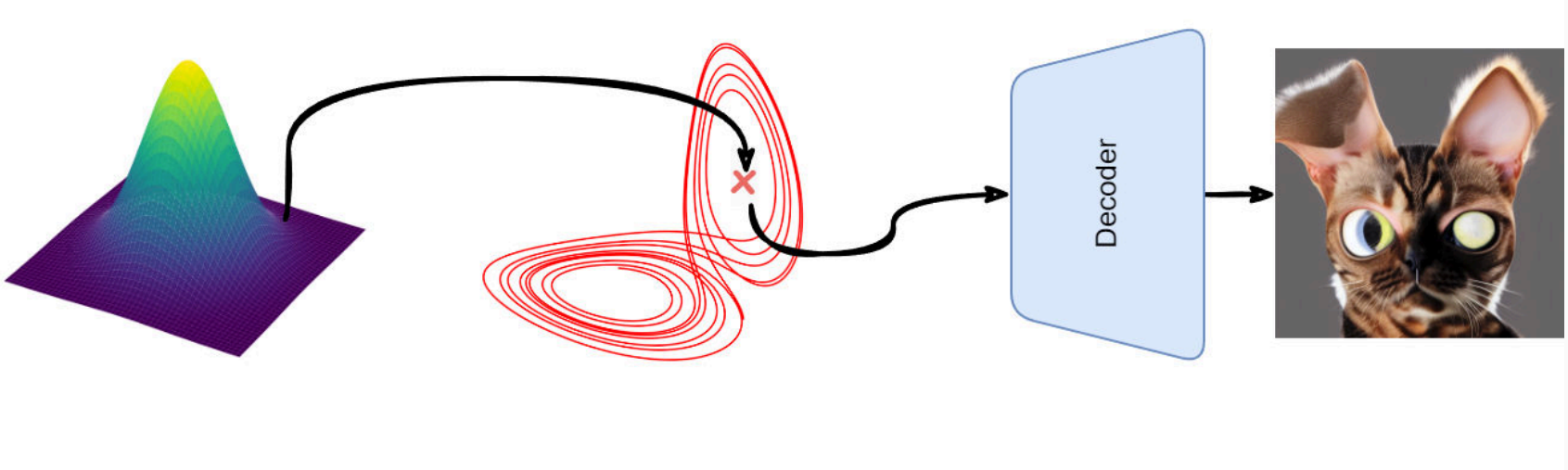
Idea and motivation

- The main task is: how do we make an autoencoder into a generative model?



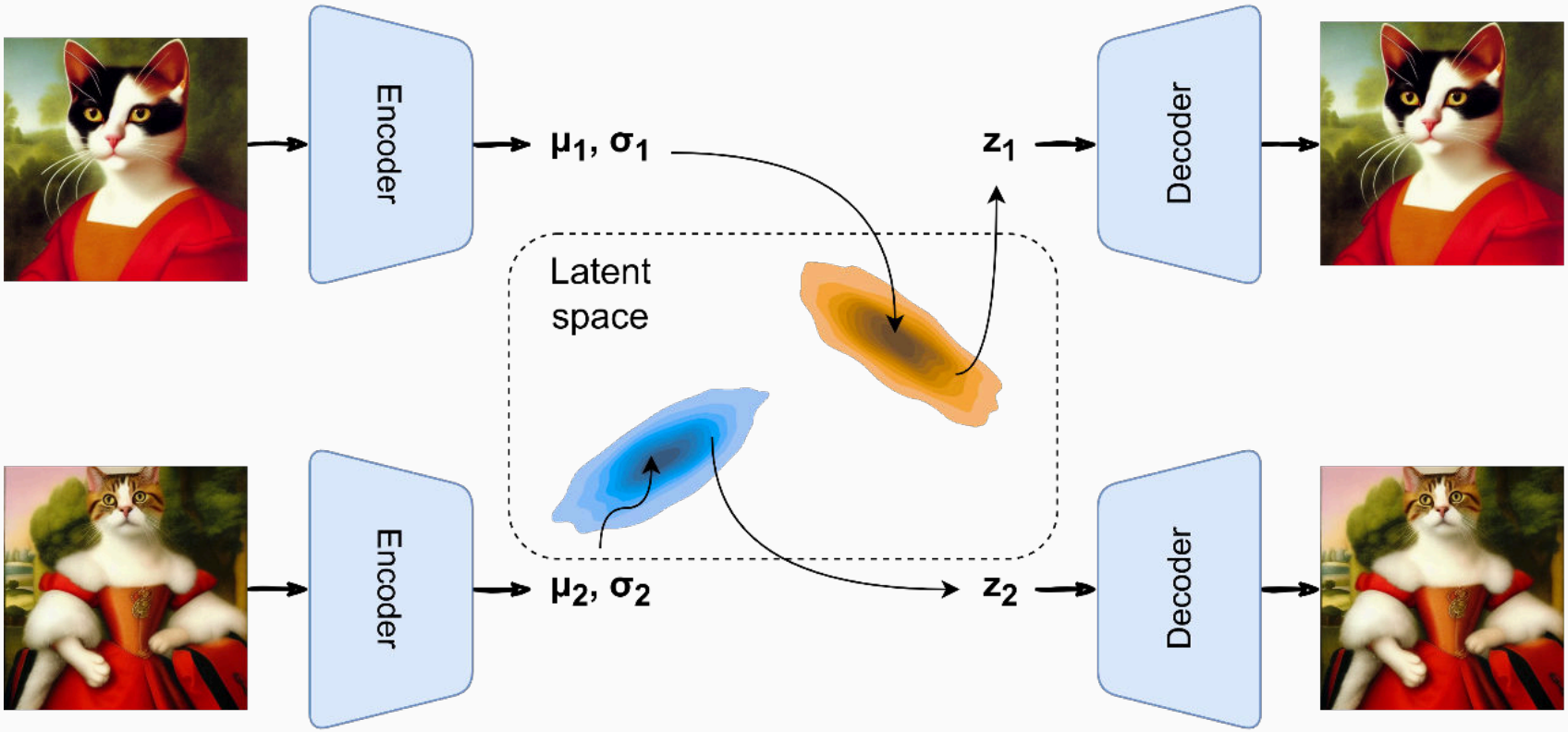
Idea and motivation

- It's hard to find a good point even in a low-dimensional space:



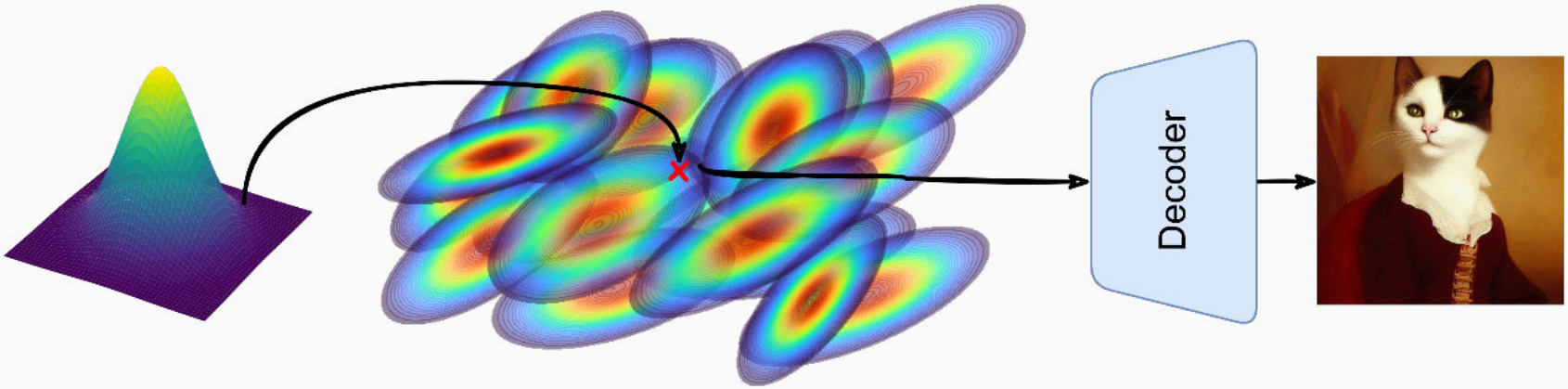
Idea and motivation

- The main idea of the variational autoencoder (Kingma et al., 2013): let every point $\mathbf{x}$ map not to a single point $\mathbf{z}$ but to an entire distribution $p(\mathbf{x}|\mathbf{z})$:



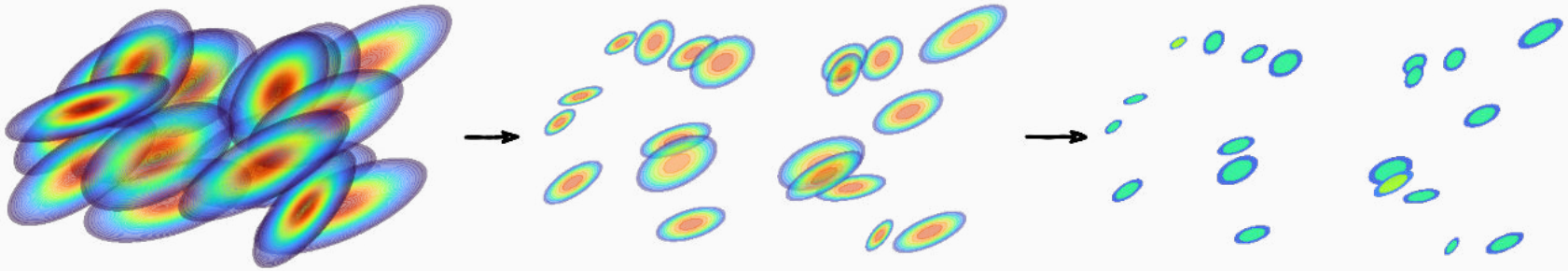
Idea and motivation

- The intuition is that the decoder has to be stable to small changes in $\mathbf{z}$; this is, in a way, an improved version of a denoising autoencoder
- We want to cover the space of latent codes with «wide enough» distributions:



Idea and motivation

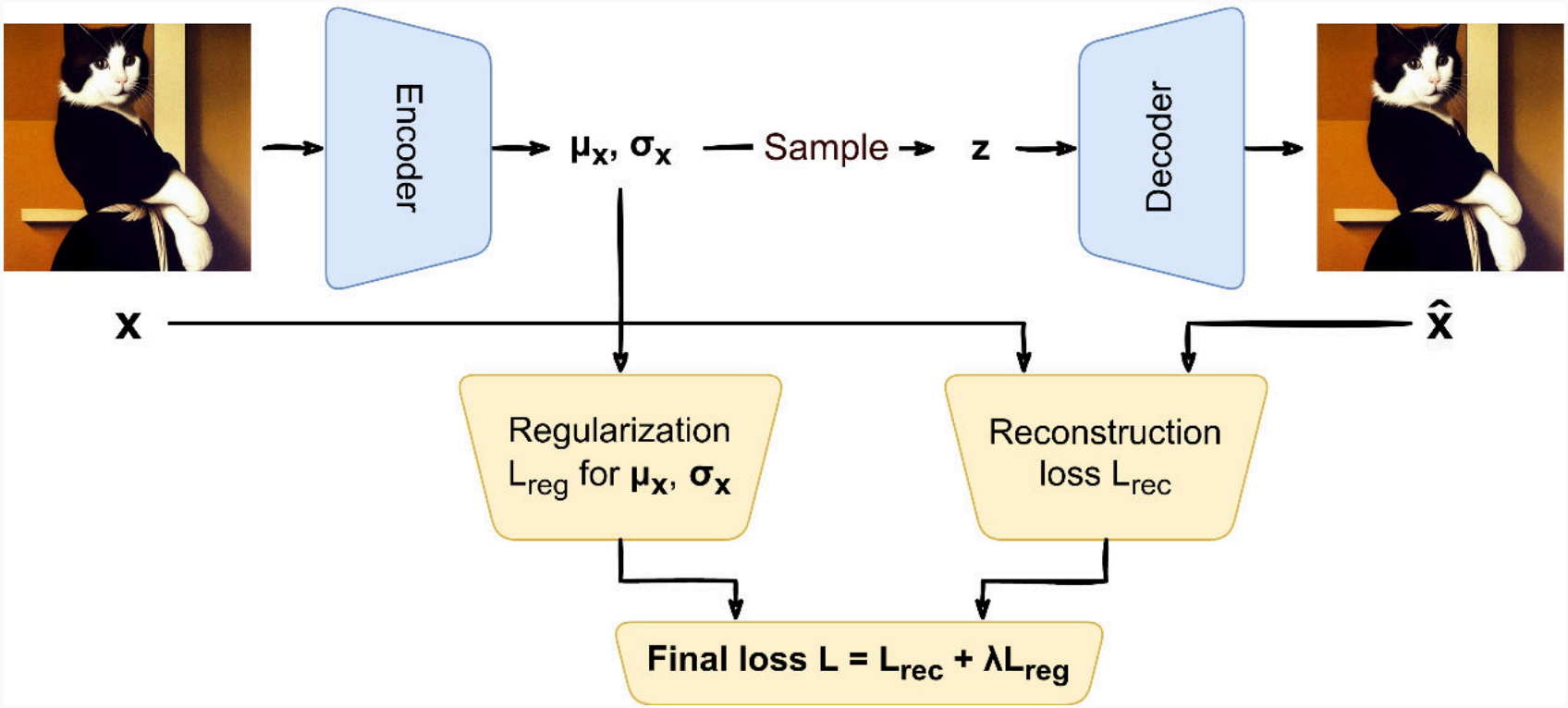
- But there is a problem: if we just train an autoencoder, it will be easier for it to decode «narrow» distributions, and we will ultimately get to distributions with zero variance, i.e., to a regular autoencoder that does not really help us



- How do we solve this problem?

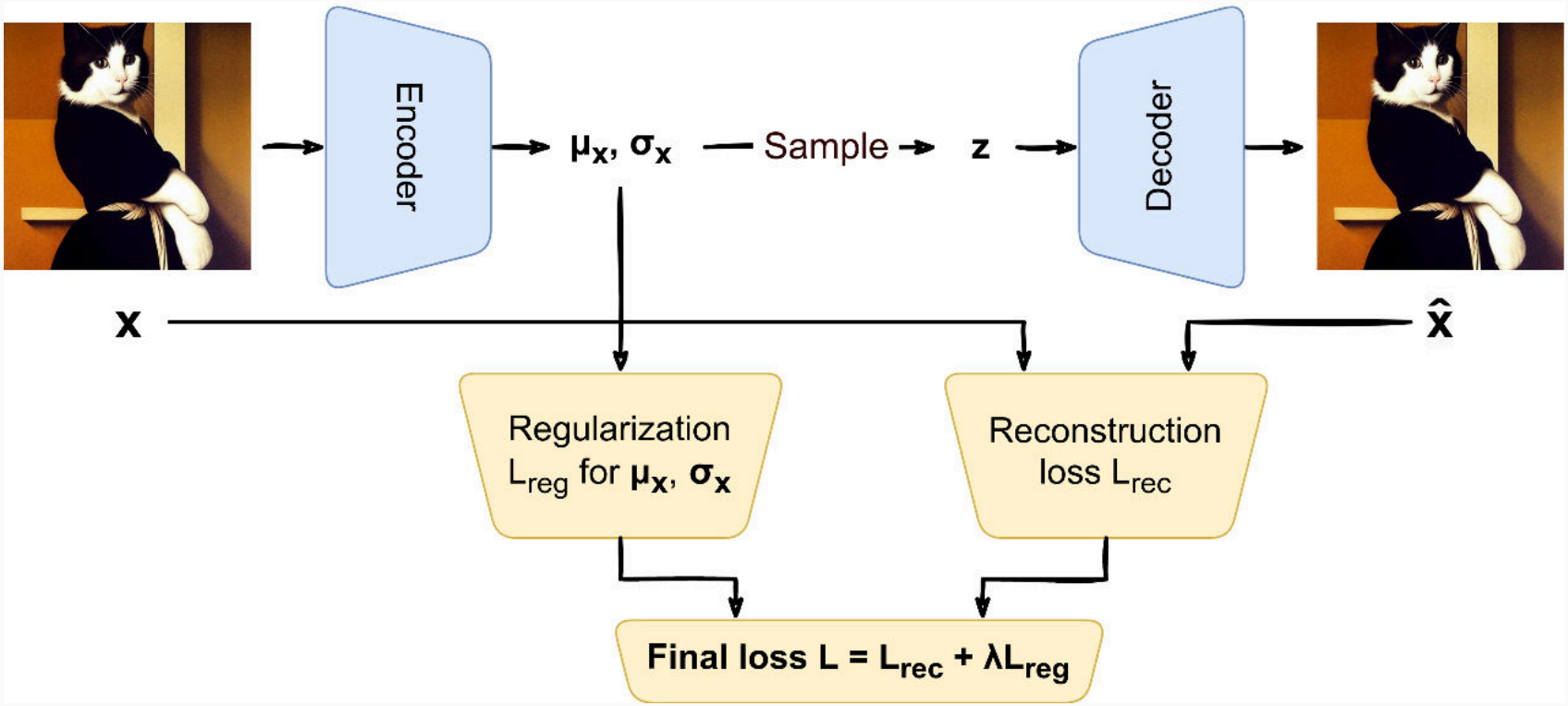
Idea and motivation

- We have to add a constraint on $p(\mathbf{x}|\mathbf{z})$, i.e., regularize it:



Idea and motivation

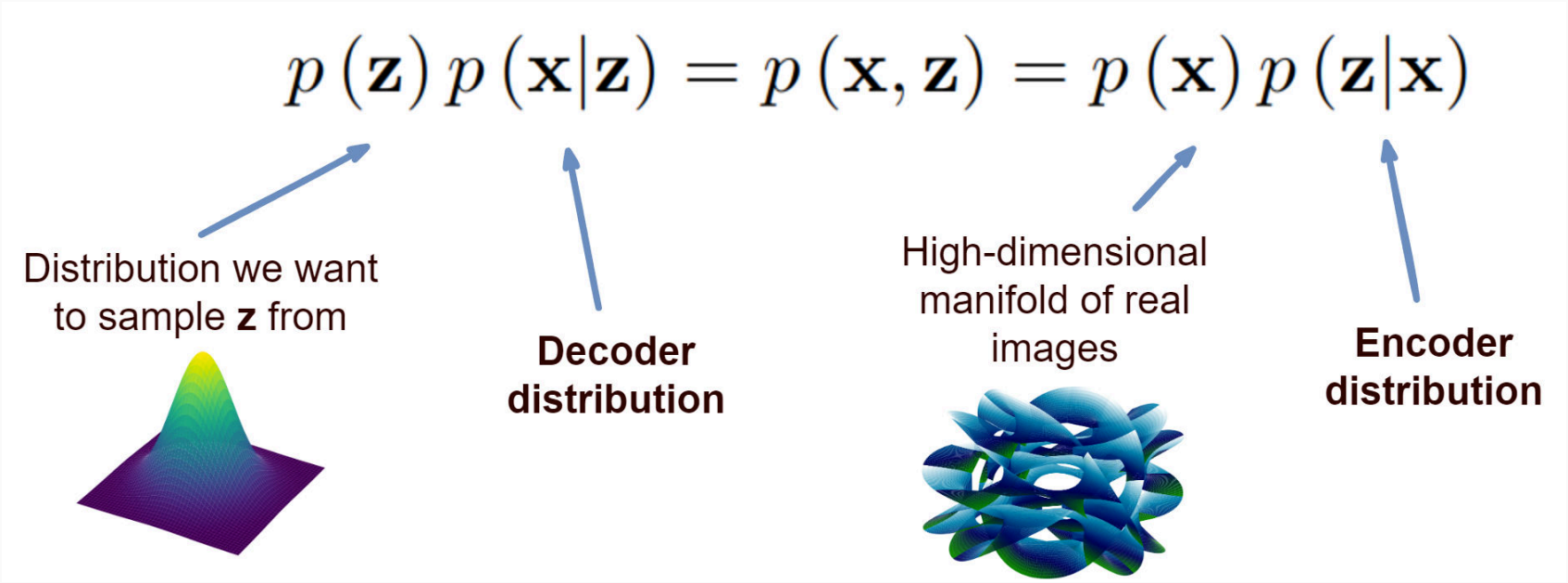
- There are now two questions left:
  - how do we choose the loss function and regularizer?
  - how can we pass gradients through the sampling process?
- Let us answer both of these questions...



Variational approximations in the VAE

- Let's start from the beginning: we want to train an encoder to find good latent codes $\mathbf{z}$ from images $\mathbf{x}$ and train a decoder that turns $\mathbf{z}$ back into $\mathbf{x}$.
- We can decompose the joint distribution $p(\mathbf{x}, \mathbf{z})$ differently:

$$p(\mathbf{z})\, p(\mathbf{x}|\mathbf{z}) = p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x})\, p(\mathbf{z}|\mathbf{x})$$

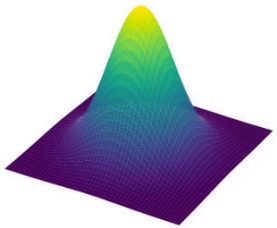


Variational approximations in the VAE

- Here:
  - $p(\mathbf{x})$ is the distribution of images,
  - $p(\mathbf{z})$ is the distribution of latent codes, i.e., a simple distribution we can sample from;
  - the other two we need to find, they are the encoder distribution $p(\mathbf{z}|\mathbf{x})$ and the decoder distribution $p(\mathbf{x}|\mathbf{z})$.

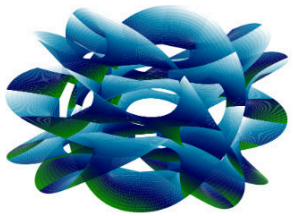
$$p(\mathbf{z})\,p(\mathbf{x}|\mathbf{z}) = p(\mathbf{x},\mathbf{z}) = p(\mathbf{x})\,p(\mathbf{z}|\mathbf{x})$$

Distribution we want to sample **z** from



Decoder distribution

High-dimensional manifold of real images



Encoder distribution

Variational approximations in the VAE

- To get a generative model we need to train both $p(\mathbf{x}|\mathbf{z})$ and $p(\mathbf{z}|\mathbf{x})$.
- But we have a simple $p(\mathbf{z})$ and a very complex $p(\mathbf{x})$!
- This means that we have to pick one:
  - either we assume that $p(\mathbf{x}|\mathbf{z})$ is simple and then look for a complex $p(\mathbf{z}|\mathbf{x})$;
  - or, vice versa, assume that $p(\mathbf{z}|\mathbf{x})$ is simple and then look for a complex $p(\mathbf{x}|\mathbf{z})$.

Variational approximations in the VAE

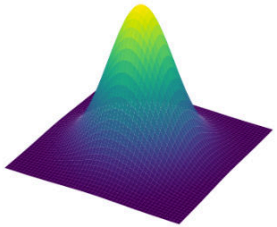
- VAE chooses the first option: suppose that

$$p(\mathbf{x}|\mathbf{z}) = N(\mathbf{x}|f(\mathbf{z}), c\mathbf{I}), \quad \text{where} \quad f(\mathbf{z}) = \text{Decoder}(\mathbf{z})$$

- On the left we have $p(\mathbf{z}|\mathbf{x})$ times $p(\mathbf{z}) = N(\mathbf{z}|0, \mathbf{I})$, i.e., a Gaussian again.

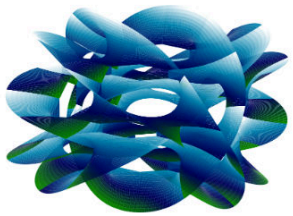
$$p(\mathbf{z})\,p(\mathbf{x}|\mathbf{z}) = p(\mathbf{x},\mathbf{z}) = p(\mathbf{x})\,p(\mathbf{z}|\mathbf{x})$$

Distribution we want to sample **z** from



Decoder distribution

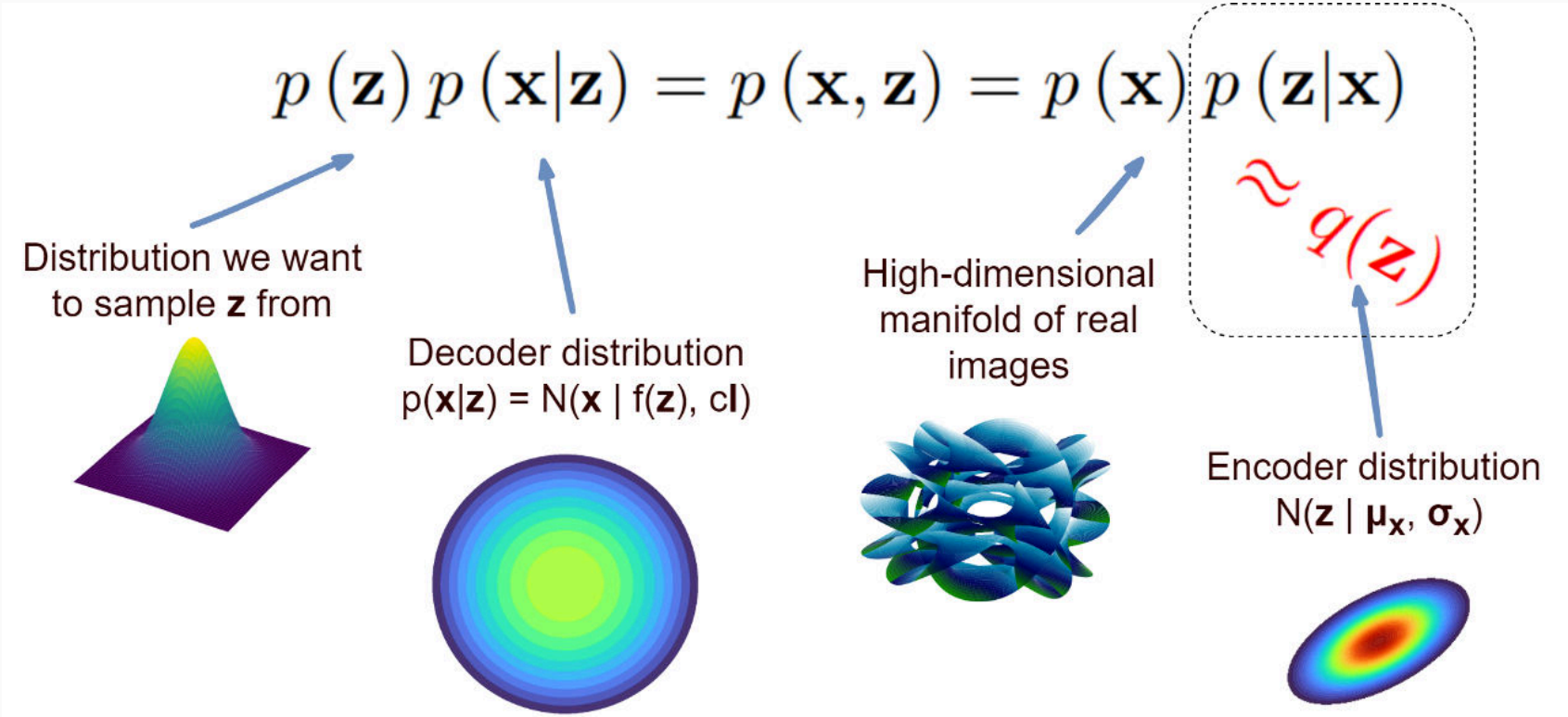
High-dimensional manifold of real images



Encoder distribution

Variational approximations in the VAE

- On the right, we will be looking for an approximation $q(\mathbf{z}) \approx p(\mathbf{z}|\mathbf{x})$:



- Namely a variational approximation...

Variational approximations in the VAE

- Let us derive the variational approximation from first principles:

$$p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x})\, p(\mathbf{z}|\mathbf{x}),$$

$$\log p(\mathbf{x}) = \log p(\mathbf{x}, \mathbf{z}) - \log p(\mathbf{z}|\mathbf{x}),$$

take the expectation over $q(\mathbf{z})$:

$$\mathbb{E}_{q(\mathbf{z})}[\log p(\mathbf{x})] = \mathbb{E}_{q(\mathbf{z})}[\log p(\mathbf{x}, \mathbf{z})] - \mathbb{E}_{q(\mathbf{z})}[\log p(\mathbf{z}|\mathbf{x})],$$

and then do standard transformations:

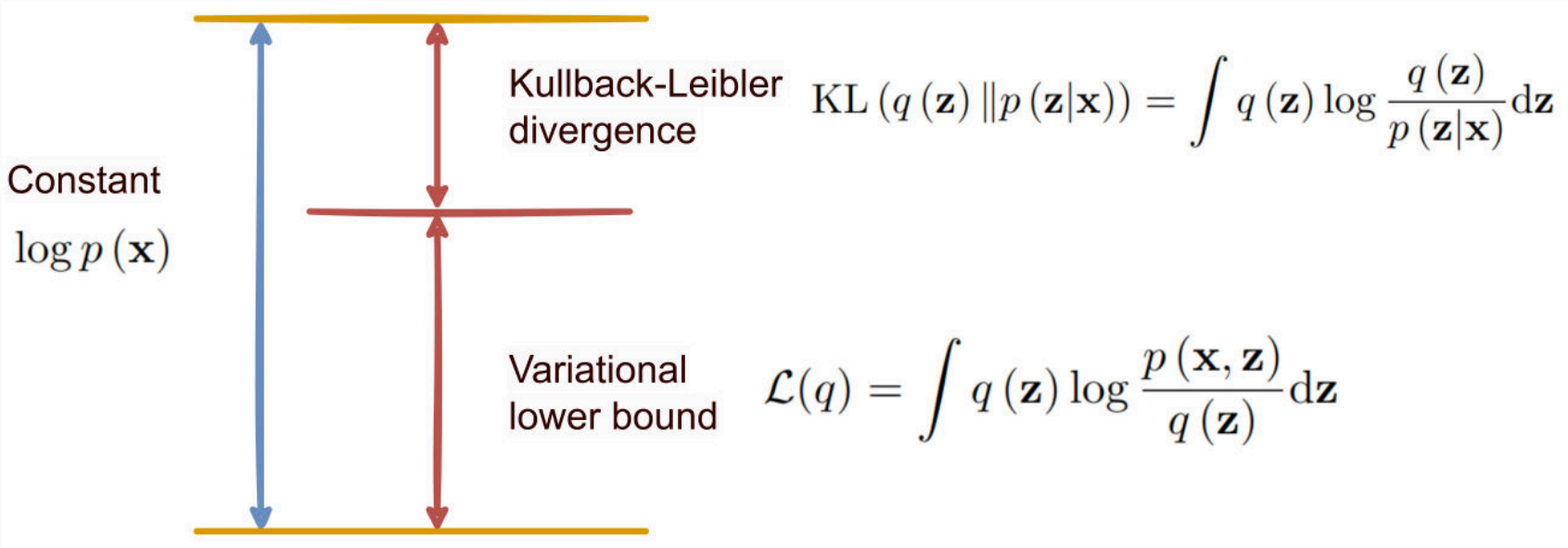
$$\begin{aligned}\log p(\mathbf{x}) =& \mathbb{E}_{q(\mathbf{z})}[\log p(\mathbf{x}, \mathbf{z})] - \mathbb{E}_{q(\mathbf{z})}[\log q(\mathbf{z})] + \\ &+ \mathbb{E}_{q(\mathbf{z})}[\log q(\mathbf{z})] - \mathbb{E}_{q(\mathbf{z})}[\log p(\mathbf{z}|\mathbf{x})], \\ \log p(\mathbf{x}) =& \int q(\mathbf{z}) \log \frac{p(\mathbf{x}, \mathbf{z})}{q(\mathbf{z})} d\mathbf{z} + \int q(\mathbf{z}) \log \frac{q(\mathbf{z})}{p(\mathbf{z}|\mathbf{x})} d\mathbf{z}.\end{aligned}$$

Variational approximations in the VAE

- We get that

$$\log p\left(\mathbf{x}\right) = L(q) + \mathrm{KL}\left(q\left(z\right) \| p\left(\mathbf{z}|\mathbf{x}\right)\right), \quad \text{where}$$

$$L(q) = \int q\left(\mathbf{z}\right) \log \frac{p\left(\mathbf{x}, \mathbf{z}\right)}{q\left(\mathbf{z}\right)} \mathrm{d}\mathbf{z}.$$



Variational approximations in the VAE

- Thus, we can approximate $p(\mathbf{z}|\mathbf{x})$ with $q(\mathbf{z})$ by maximizing $L(q)$. In the case of VAE it is easy to find:

$$
\begin{aligned}
L(q) &= \int q(\mathbf{z}) \log \frac{p(\mathbf{x}, \mathbf{z})}{q(\mathbf{z})} \mathrm{d}\mathbf{z} = \int q(\mathbf{z}) \log \frac{p(\mathbf{z})\, p(\mathbf{x}|\mathbf{z})}{q(\mathbf{z})} \mathrm{d}\mathbf{z} \\
&= \int q(\mathbf{z}) \log p(\mathbf{x}|\mathbf{z}) \mathrm{d}\mathbf{z} + \int q(\mathbf{z}) \log \frac{p(\mathbf{z})}{q(\mathbf{z})} \mathrm{d}\mathbf{z} \\
&= \int q(\mathbf{z}) \log N(\mathbf{x}|f(\mathbf{z}), c\mathbf{I}) \mathrm{d}\mathbf{z} - \mathrm{KL}(q(\mathbf{z}) \| p(\mathbf{z})) \\
&= -\frac{1}{2c} \mathbb{E}_{q(\mathbf{z})} \left[\|\mathbf{x} - f(\mathbf{z})\|^2 \right] - \mathrm{KL}(q(\mathbf{z}) \| p(\mathbf{z})) .
\end{aligned}
$$

Variational approximations in the VAE

- And thus we have arrived at exactly what our intuition predicted:

$$-\mathcal{L}(q) = \frac{1}{2c}\mathbb{E}_{q(\mathbf{z})}\left[\|\mathbf{x} - f(\mathbf{z})\|^2\right] + \mathrm{KL}\left(q\left(\mathbf{z}\right)\|p\left(\mathbf{z}\right)\right)$$

Reconstruction loss L_{rec}

Regularization L_{reg} for μ_X, σ_X

Final loss
$L = L_{rec} + \lambda L_{reg}$

Variational approximations in the VAE

- But now we know exactly what L_{rec} looks like, and it is also easy to find the KL divergence between two Gaussians:

$$\mathrm{KL}\left(q\left(\mathbf{z}\right)\|p\left(\mathbf{z}\right)\right)=\frac{1}{2}\sum_{j=1}^{d}\left(\sigma_{\mathbf{x},j}^{2}+\mu_{\mathbf{x},j}^{2}-\log\sigma_{\mathbf{x},j}^{2}-1\right).$$

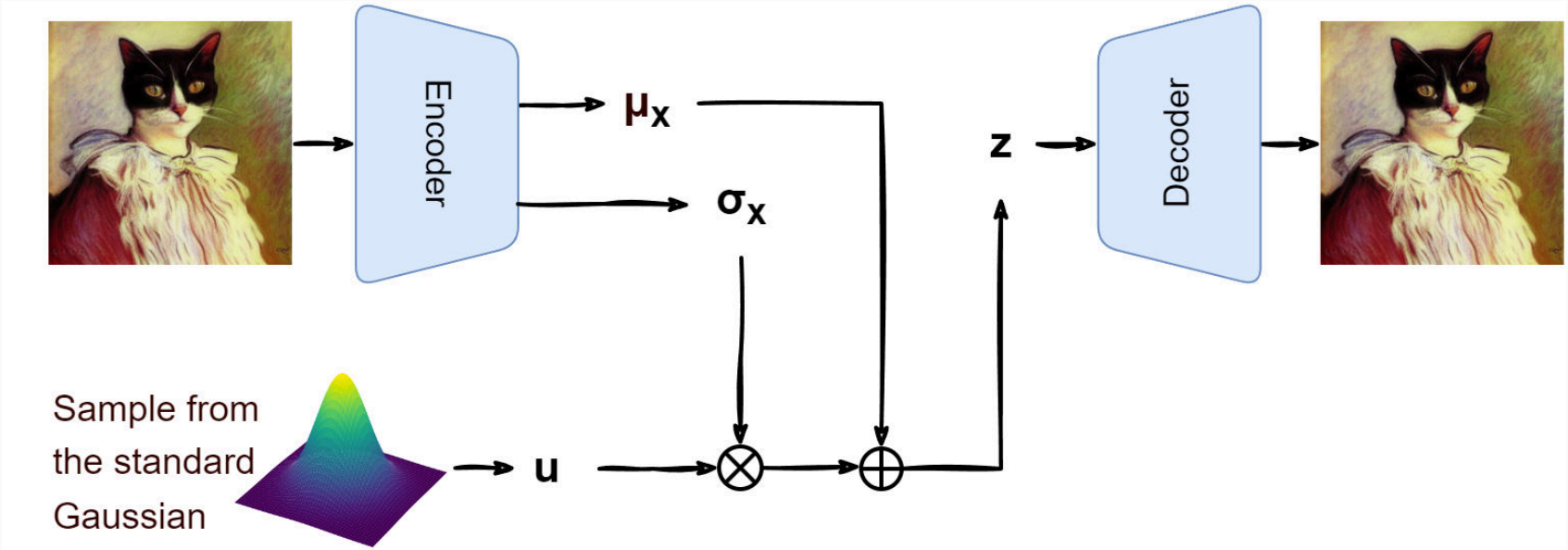
- This leads to the final loss function for the variational autoencoder:

$$L(q)=-\frac{1}{2c}\mathbb{E}_{q(\mathbf{z})}\left[\|\mathbf{x}-f(\mathbf{z})\|^{2}\right]-\frac{1}{2}\sum_{j=1}^{d}\left(\sigma_{\mathbf{x},j}^{2}+\mu_{\mathbf{x},j}^{2}-\log\sigma_{\mathbf{x},j}^{2}-1\right).$$

- It only remains to solve the sampling problem...

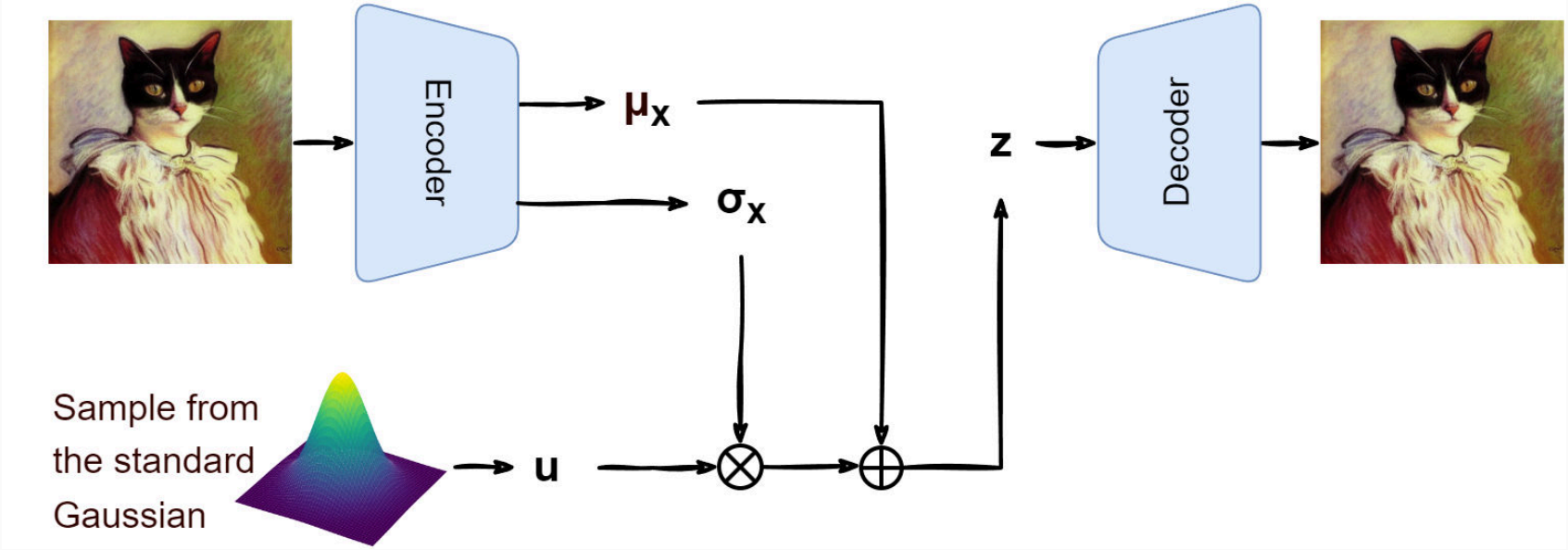
Reparametrization trick

- To train the model, we need to pass the gradients through $\mathbf{z}$ back to the encoder, but we can't go through sampling
- Reparametrization trick: let us first sample and then transform the result



Reparametrization trick

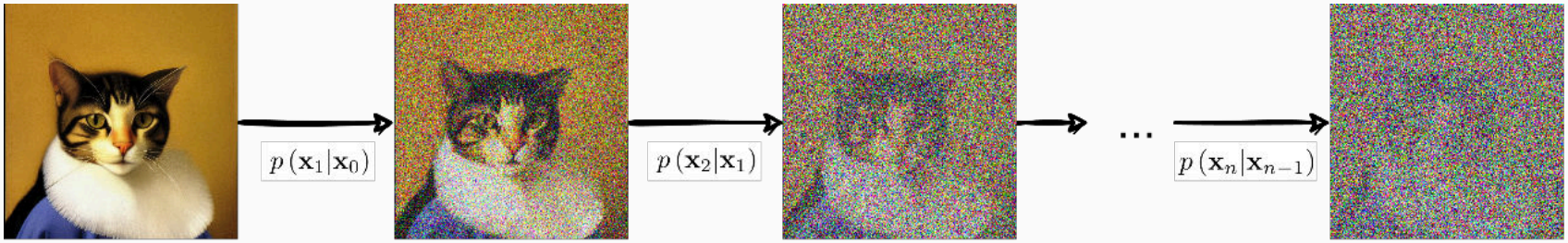
- Now we have all components firmly in place: we sample a mini-batch of $\mathbf{u}$ for an input mini-batch of $\mathbf{x}$ and use it for training



Diffusion models

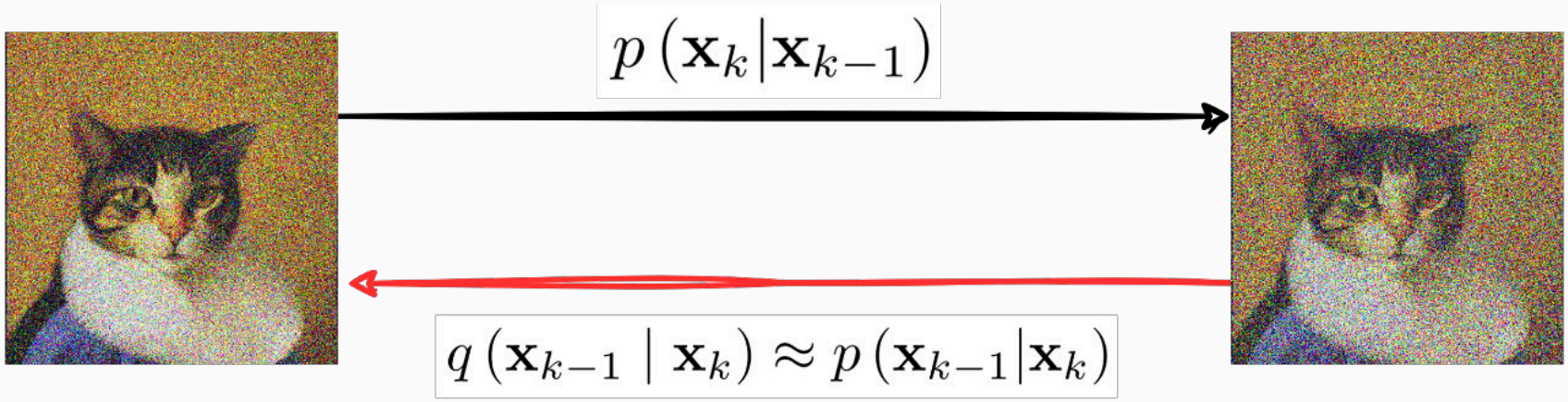
Diffusion models

- (Sohl-Dickstein et al., 2015): Deep Unsupervised Learning using Nonequilibrium Thermodynamics
- An idea from statistical physics: let us train a Markov chain that will gradually turn random noise into the distribution we want
- To do that, let us start with a chain that turns the distribution we want into random noise:

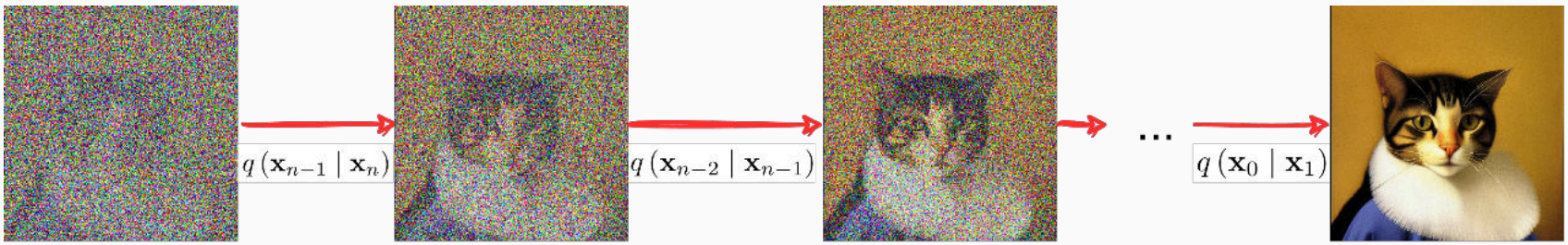


Diffusion models

- And then let us try to reverse each step:



- If we succeed, then later we can start from random noise and move backwards:



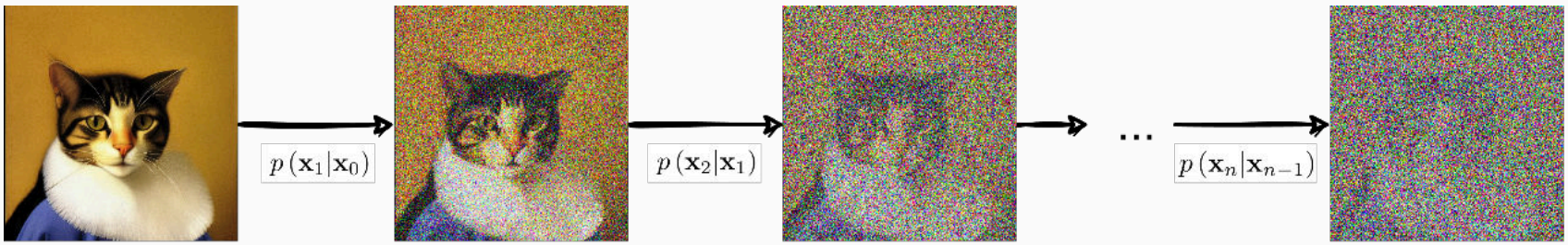
Diffusion models

- Forward diffusion: let us start with $p(\mathbf{x}_0)$ and add Gaussian noise:

$$q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) = N\left(\mathbf{x}_t \middle| \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \mathbf{I}\right).$$

- Then $\mathbf{x}_0$ gradually turns into noise, say, in T steps:

$$q(\mathbf{x}_{1:T} \mid \mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t \mid \mathbf{x}_{t-1}).$$



- And for each t we can sample $\mathbf{x}_t$ in closed form via the reparametrization trick: let $\epsilon \sim N(0, \mathbf{I})$ and denote $\alpha_t = 1 - \beta_t$, $A_t = \prod_{i=1}^{T} \alpha_i$; then

$$\mathbf{x}_t = \sqrt{\alpha_t}\mathbf{x}_{t-1} + \sqrt{1-\alpha_t}\epsilon = \sqrt{\alpha_t\alpha_{t-1}}\mathbf{x}_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\epsilon = \ldots$$

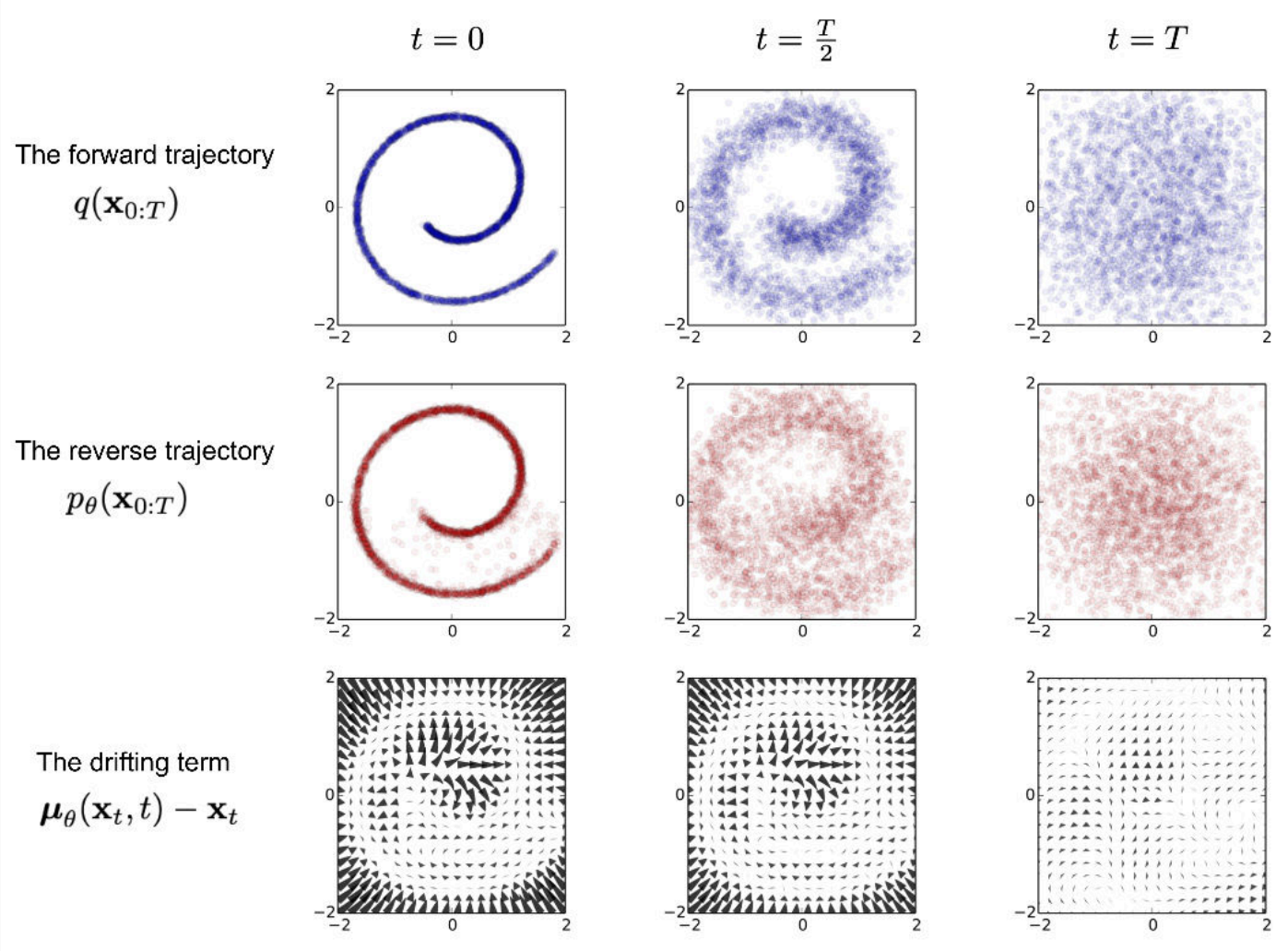
$$\ldots = \sqrt{A_t}\mathbf{x}_0 + \sqrt{1-A_t}\epsilon, \text{ that is } q(\mathbf{x}_t \mid \mathbf{x}_0) = N\left(\mathbf{x}_t \middle| \sqrt{A_t}\mathbf{x}_0, (1-A_t)\mathbf{I}\right),$$

because when we add two Gaussians $N(0, \sigma_1^2\mathbf{I})$ and $N(0, \sigma_1^2\mathbf{I})$, we get a Gaussian $N(0, (\sigma_1^2 + \sigma_2^2)\mathbf{I})$, and in our case this is

$$(1-\alpha_t) + \alpha_t(1-\alpha_t) = 1 - \alpha_t\alpha_{t-1}.$$

Diffusion models

- And now the task, it would seem, is to reverse this process, to learn how to sample from $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$



Diffusion models

- For small β_t the reverse distributions will also be Gaussian, but their parameters now depend on the data points.
- Reverse diffusion: now we need to train the distributions

$$p_\theta\left(\mathbf{x}_{0:T}\right) = p_\theta\left(\mathbf{x}_T\right) \prod_{t=1}^{T} p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right), \quad \text{where}$$

$$p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right) = N\left(\mathbf{x}_{t-1} \middle| \mu_\theta(\mathbf{x}_t, t), \Sigma_\theta(\mathbf{x}_t, t)\right).$$

- If we knew $\mathbf{x}_0$, we could reverse it analytically, by the way:

$$q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0\right) = q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{x}_0\right) \frac{q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_0\right)}{q\left(\mathbf{x}_t \mid \mathbf{x}_0\right)} \propto$$

$$\propto e^{-\frac{1}{2}\left(\frac{\left(\mathbf{x}_t - \sqrt{\alpha_t}\mathbf{x}_{t-1}\right)^2}{\beta_t} + \frac{\left(\mathbf{x}_{t-1} - \sqrt{A_{t-1}}\mathbf{x}_0\right)^2}{1-A_{t-1}} + \frac{\left(\mathbf{x}_t - \sqrt{A_t}\mathbf{x}_0\right)^2}{1-A_t}\right)},$$

and in this formula we now need to complete the square in $\mathbf{x}_{t-1}$...

- This yields

$$q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0\right) = N\left(\mathbf{x}_{t-1} \middle| \tilde{\mu}\left(\mathbf{x}_t, \mathbf{x}_0\right), \tilde{\beta}_t \mathbf{I}\right), \quad \text{where}$$

$$\tilde{\beta}_t = \frac{1 - A_{t-1}}{1 - A_t} \cdot \beta_t,$$

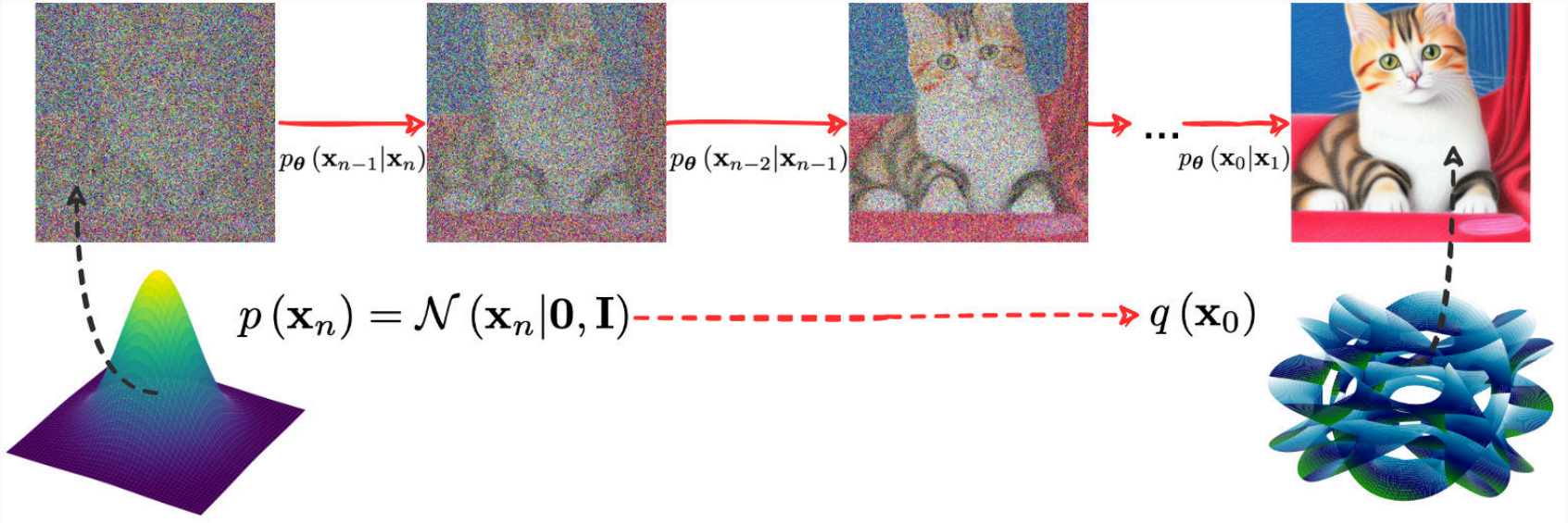
$$\tilde{\mu}\left(\mathbf{x}_t, \mathbf{x}_0\right) = \frac{\sqrt{\alpha_t}\left(1 - A_{t-1}\right)}{1 - A_t}\mathbf{x}_t + \frac{\sqrt{A_{t-1}}\beta_t}{1 - A_t}\mathbf{x}_0,$$

and since we know that $\mathbf{x}_t = \sqrt{A_t}\mathbf{x}_0 + \sqrt{1 - A_t}\epsilon$,

$$\tilde{\mu}\left(\mathbf{x}_t, \mathbf{x}_0\right) = \frac{1}{\sqrt{A_t}}\left(\mathbf{x}_t - \frac{1 - \alpha_t}{\sqrt{1 - A_t}}\epsilon_t\right).$$

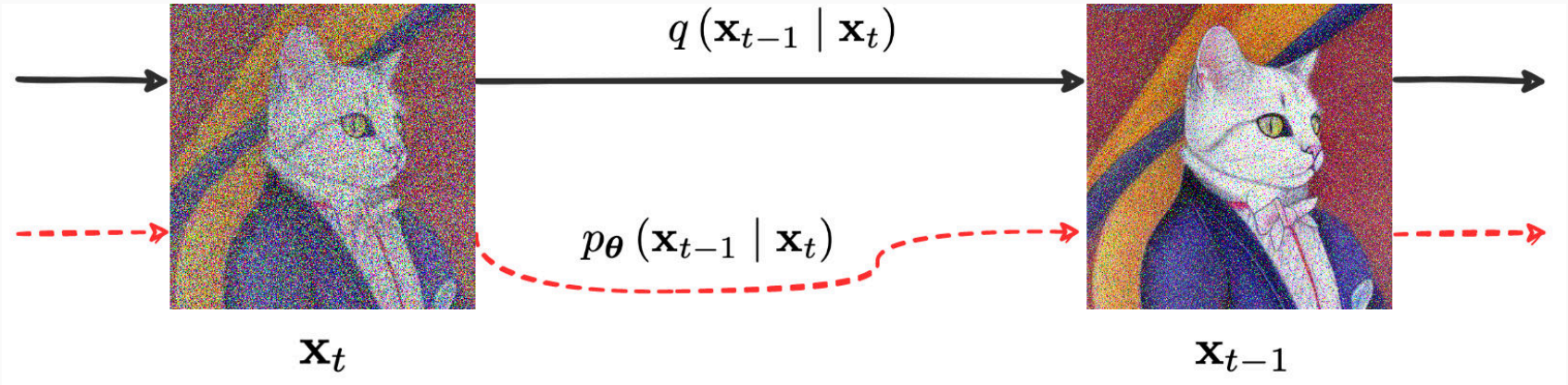
Diffusion models

- Ultimately we want the reverse diffusion to reconstruct $q(\mathbf{x}_0)$ from a standard input into $\mathbf{x}_n$; roughly like this:



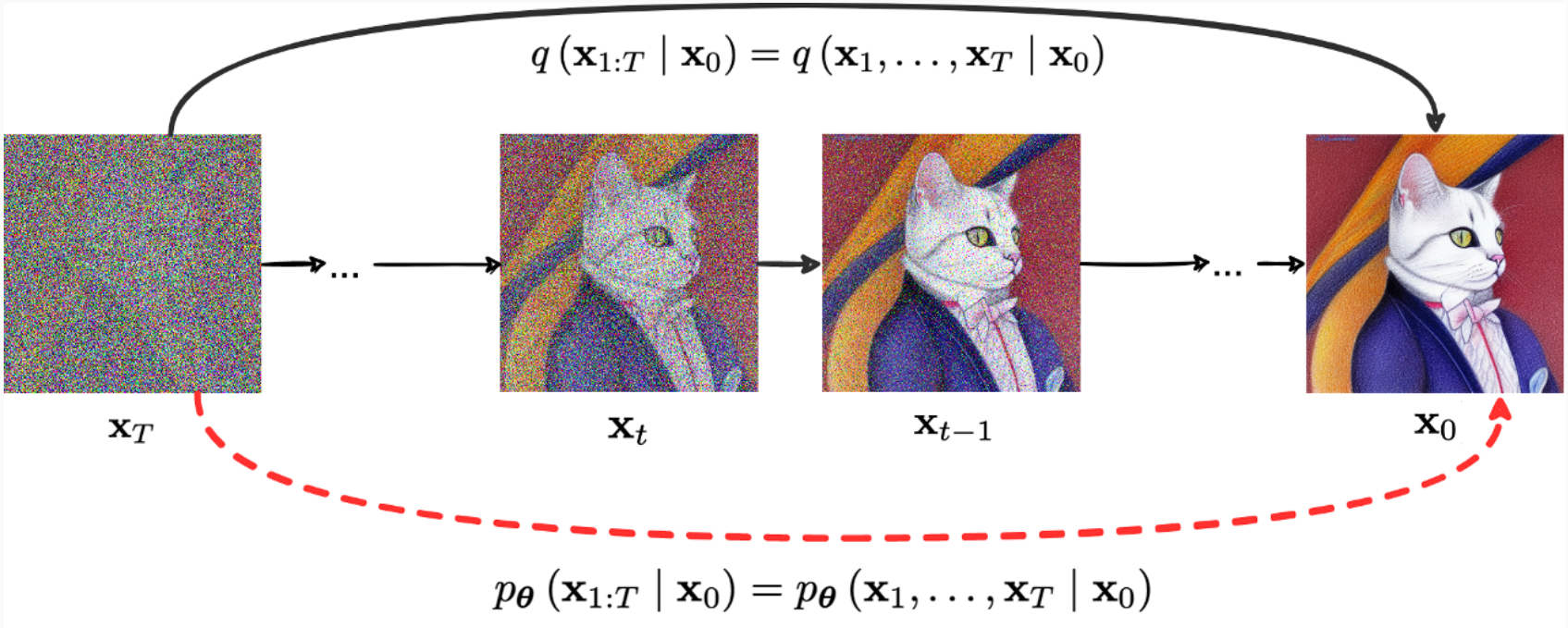
Diffusion models

- What should we do? As always in Bayesian inference, approximate!
- At each step the model should become a good approximation to $q(\mathbf{x}_t \mid \mathbf{x}_{t-1})$, without conditioning on the unknown $\mathbf{x}_0$:



Diffusion models

- To obtain this approximation, we need a variational lower bound, similar to the one we had in VAE and DALL-E
- We start with a bound on the global distribution $q\left(\mathbf{x}_{1:T} \mid \mathbf{x}_0\right) = q\left(\mathbf{x}_1, \ldots, \mathbf{x}_T \mid \mathbf{x}_0\right)$:



- And then it will decompose into bounds for the individual steps of the diffusion process

- Let us recall the idea of variational approximations:

$$p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x})\, p(\mathbf{z}|\mathbf{x}),$$

$$\log p(\mathbf{x}) = \log p(\mathbf{x}, \mathbf{z}) - \log p(\mathbf{z}|\mathbf{x}),$$

let us take the expectation over $q(\mathbf{z})$:

$$\mathbb{E}_{q(\mathbf{z})}[\log p(\mathbf{x})] = \mathbb{E}_{q(\mathbf{z})}[\log p(\mathbf{x}, \mathbf{z})] - \mathbb{E}_{q(\mathbf{z})}[\log p(\mathbf{z}|\mathbf{x})],$$

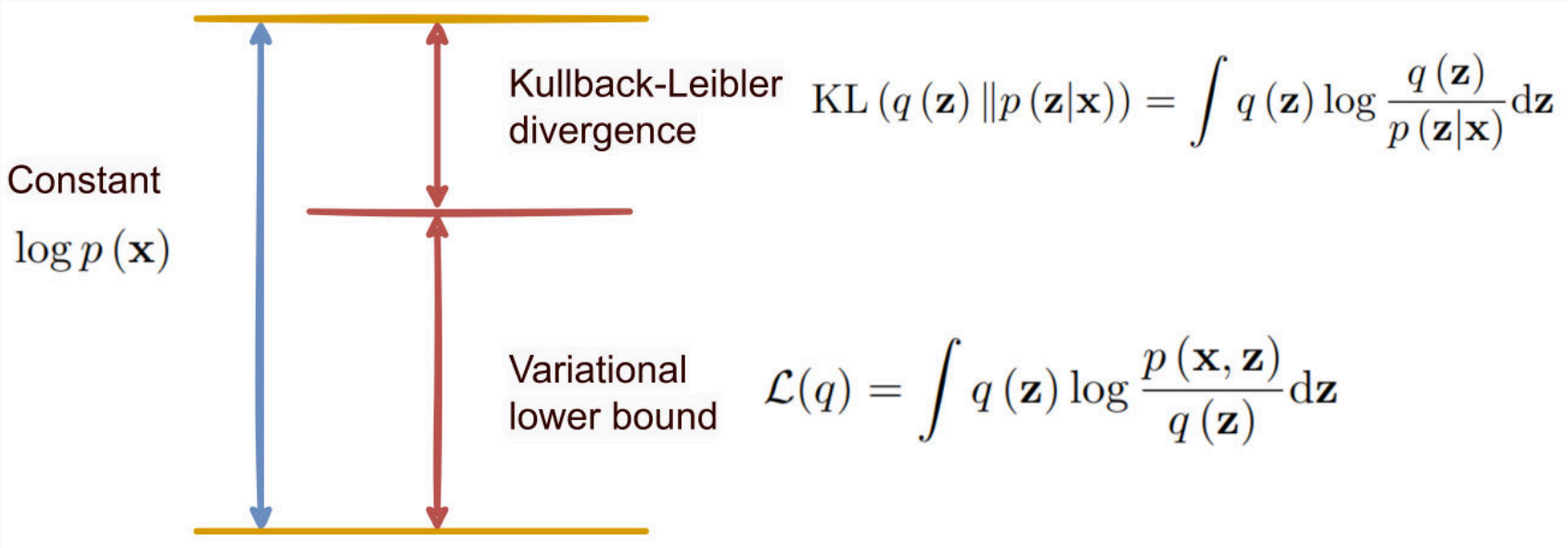
and then do the standard manipulations:

$$\begin{aligned}\log p(\mathbf{x}) =& \mathbb{E}_{q(\mathbf{z})}[\log p(\mathbf{x}, \mathbf{z})] - \mathbb{E}_{q(\mathbf{z})}[\log q(\mathbf{z})] + \\ &+ \mathbb{E}_{q(\mathbf{z})}[\log q(\mathbf{z})] - \mathbb{E}_{q(\mathbf{z})}[\log p(\mathbf{z}|\mathbf{x})], \\ \log p(\mathbf{x}) =& \int q(\mathbf{z}) \log \frac{p(\mathbf{x}, \mathbf{z})}{q(\mathbf{z})} d\mathbf{z} + \int q(\mathbf{z}) \log \frac{q(\mathbf{z})}{p(\mathbf{z}|\mathbf{x})} d\mathbf{z}.\end{aligned}$$

- In the end it turns out that

$$\log p\left(\mathbf{x}\right) = L(q) + \mathrm{KL}\left(q\left(\mathbf{z}\right) \| p\left(\mathbf{z}|\mathbf{x}\right)\right), \quad \text{where}$$

$$L(q) = \int q\left(\mathbf{z}\right) \log \frac{p\left(\mathbf{x}, \mathbf{z}\right)}{q\left(\mathbf{z}\right)} \mathrm{d}\mathbf{z}.$$



- Let us derive the variational bound for the diffusion model again from first principles:

$$\log p_\theta(\mathbf{x}_0) = \log p_\theta(\mathbf{x}_0, \dots, \mathbf{x}_T) - \log p_\theta(\mathbf{x}_1, \dots, \mathbf{x}_T \mid \mathbf{x}_0),$$

$$\mathbb{E}_{q(\mathbf{x}_0)}[\log p_\theta(\mathbf{x}_0)] = \mathbb{E}_{q(\mathbf{x}_{0:T})}[\log p_\theta(\mathbf{x}_{0:T})] - \mathbb{E}_{q(\mathbf{x}_{0:T})}[\log p_\theta(\mathbf{x}_{1:T} \mid \mathbf{x}_0)] =$$
$$= \mathbb{E}_{q(\mathbf{x}_{0:T})}\left[\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)}\right] + \mathbb{E}_{q(\mathbf{x}_{0:T})}\left[\log \frac{q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)}{p_\theta(\mathbf{x}_{1:T} \mid \mathbf{x}_0)}\right].$$

- Here the second term is the KL divergence, and we get the loss function as minus the variational lower bound:

$$L = \mathbb{E}_{q(\mathbf{x}_{0:T})}\left[\log \frac{q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})}\right] \geq -\mathbb{E}_{q(\mathbf{x}_0)}[\log p_\theta(\mathbf{x}_0)].$$

- This bound can be computed as a sum:

$$
\begin{aligned}
L =& \mathbb{E}_q \left[\log \frac{q\left(\mathbf{x}_{1:T} \mid \mathbf{x}_0\right)}{p_\theta\left(\mathbf{x}_{0:T}\right)} \right] = \mathbb{E}_q \left[\log \frac{\prod_{t=1}^{T} q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}\right)}{p_\theta\left(\mathbf{x}_T\right) \prod_{t=1}^{T} p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)} \right] \\
=& \mathbb{E}_q \left[-\log p_\theta\left(\mathbf{x}_T\right) + \sum_{t=1}^{T} \log \frac{q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}\right)}{p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)} \right] \\
=& \mathbb{E}_q \left[-\log p_\theta\left(\mathbf{x}_T\right) + \sum_{t=2}^{T} \log \frac{q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}\right)}{p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)} + \log \frac{q\left(\mathbf{x}_1 \mid \mathbf{x}_0\right)}{p_\theta\left(\mathbf{x}_0 \mid \mathbf{x}_1\right)} \right] \\
=& \mathbb{E}_q \left[-\log p_\theta\left(\mathbf{x}_T\right) + \sum_{t=2}^{T} \log \left(\frac{q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{x}_0\right)}{p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)} \frac{q\left(\mathbf{x}_t \mid \mathbf{x}_0\right)}{q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_0\right)} \right) + \log \frac{q\left(\mathbf{x}_1 \mid \mathbf{x}_0\right)}{p_\theta\left(\mathbf{x}_0 \mid \mathbf{x}_1\right)} \right] \\
=& \mathbb{E}_q \left[-\log p_\theta\left(\mathbf{x}_T\right) + \sum_{t=2}^{T} \log \frac{q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{x}_0\right)}{p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)} + \log \frac{q\left(\mathbf{x}_T \mid \mathbf{x}_0\right)}{q\left(\mathbf{x}_1 \mid \mathbf{x}_0\right)} + \log \frac{q\left(\mathbf{x}_1 \mid \mathbf{x}_0\right)}{p_\theta\left(\mathbf{x}_0 \mid \mathbf{x}_1\right)} \right] \\
=& \mathbb{E}_q \left[\log \frac{q\left(\mathbf{x}_T \mid \mathbf{x}_0\right)}{p_\theta\left(\mathbf{x}_T\right)} + \sum_{t=2}^{T} \log \frac{q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{x}_0\right)}{p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)} - \log p_\theta\left(\mathbf{x}_0 \mid \mathbf{x}_1\right) \right].
\end{aligned}
$$

- In the end we obtain

$$
\begin{aligned}
L &= L_T + L_{T-1} + \ldots + L_0, \quad \text{where} \\
L_T &= \mathrm{KL}\left(q\left(\mathbf{x}_T \mid \mathbf{x}_0\right) \| p_\theta\left(\mathbf{x}_T\right)\right), \\
L_t &= \mathrm{KL}\left(q\left(\mathbf{x}_t \mid \mathbf{x}_{t+1}, \mathbf{x}_0\right) \| p_\theta\left(\mathbf{x}_t \mid \mathbf{x}_{t+1}\right)\right), \quad t = 1, \ldots, T-1, \\
L_0 &= -\log p_\theta\left(\mathbf{x}_0 \mid \mathbf{x}_1\right).
\end{aligned}
$$

- These are all KL divergences between Gaussians, which can be computed and substituted into the loss function
- For example, in L_t we use the Gaussian parametrization

$$p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right) = N\left(\mathbf{x}_{t-1} \middle| \mu_\theta(\mathbf{x}_t, t), \Sigma_\theta(\mathbf{x}_t, t)\right)$$

and try to match its parameters with $q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{x}_0\right)$. For the mean, for example,

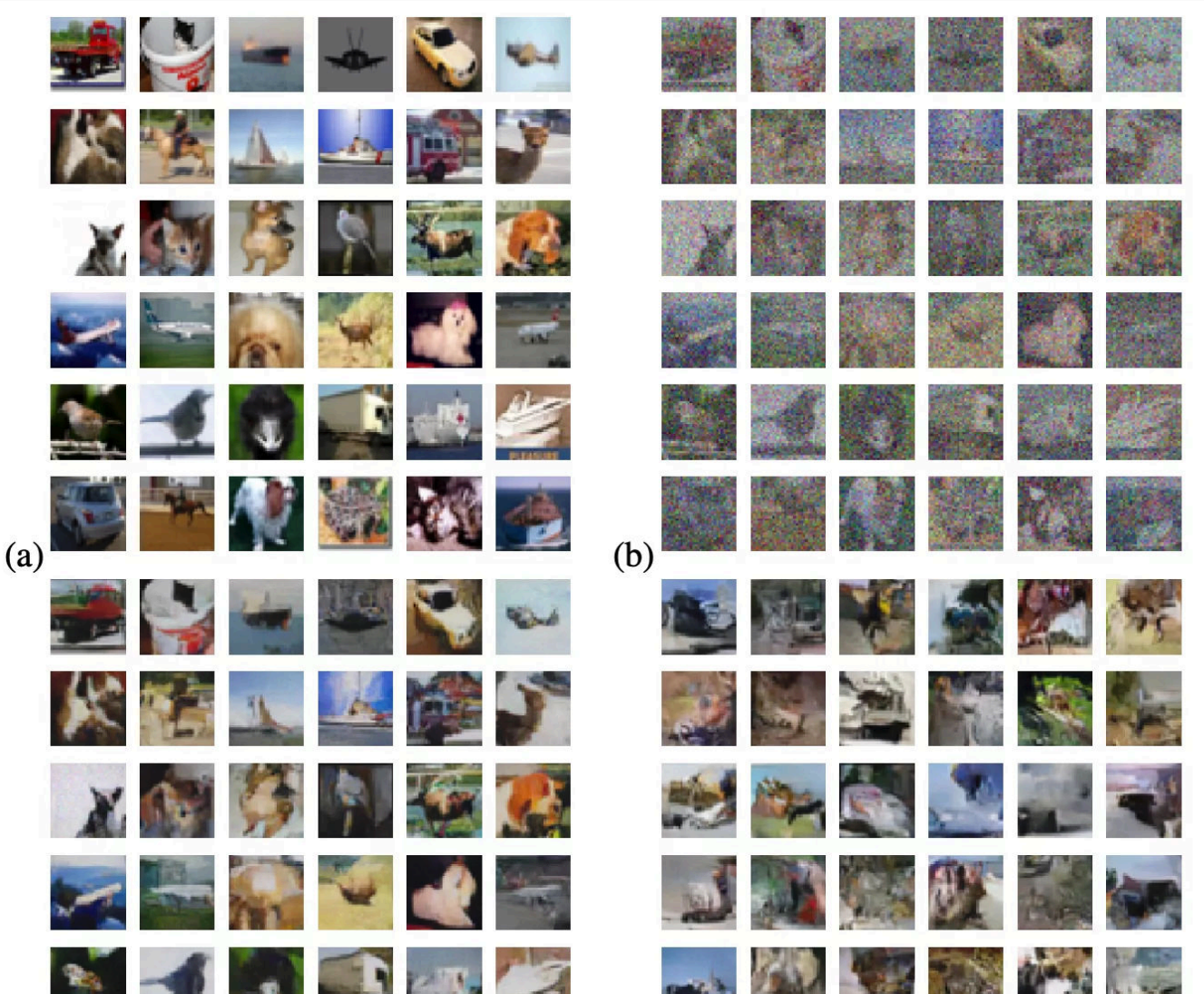
$$\mu_\theta(\mathbf{x}_t, t) \approx \tilde{\mu}_t\left(\mathbf{x}_t, \mathbf{x}_0\right) = \frac{1}{\sqrt{A_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-A_t}}\epsilon_t\right),$$

and since we know $\mathbf{x}_t$ during training, we can parametrize the noise directly:

$$p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right) = N\left(\mathbf{x}_{t-1} \middle| \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-A_t}}\epsilon_\theta(\mathbf{x}_t, t)\right), \Sigma_\theta(\mathbf{x}_t, t)\right).$$

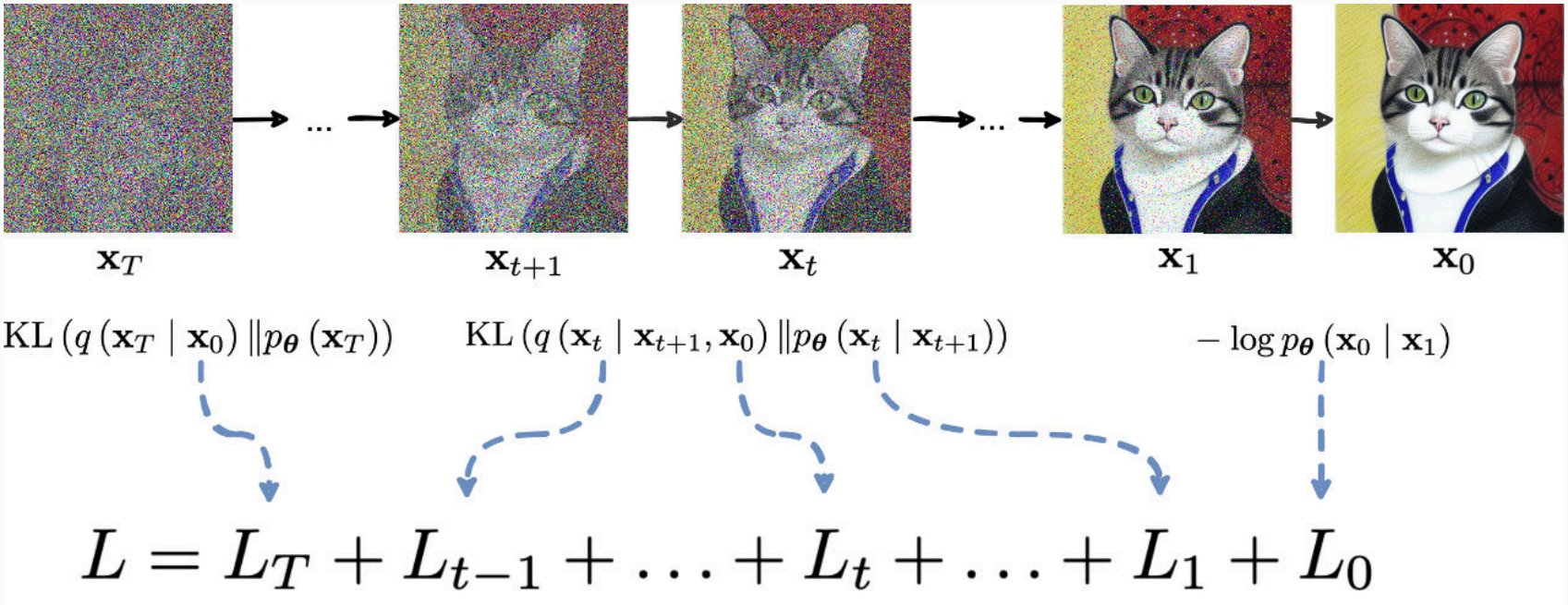
Diffusion models

- (Sohl-Dickstein et al., 2015) on its own did not work too well, except perhaps on CIFAR:



DDPM

- The next step: “Denoising Diffusion Probabilistic Models” (DDPM; Ho et al., 2020)
- The same basic idea, the same structure of the loss function:



DDPM

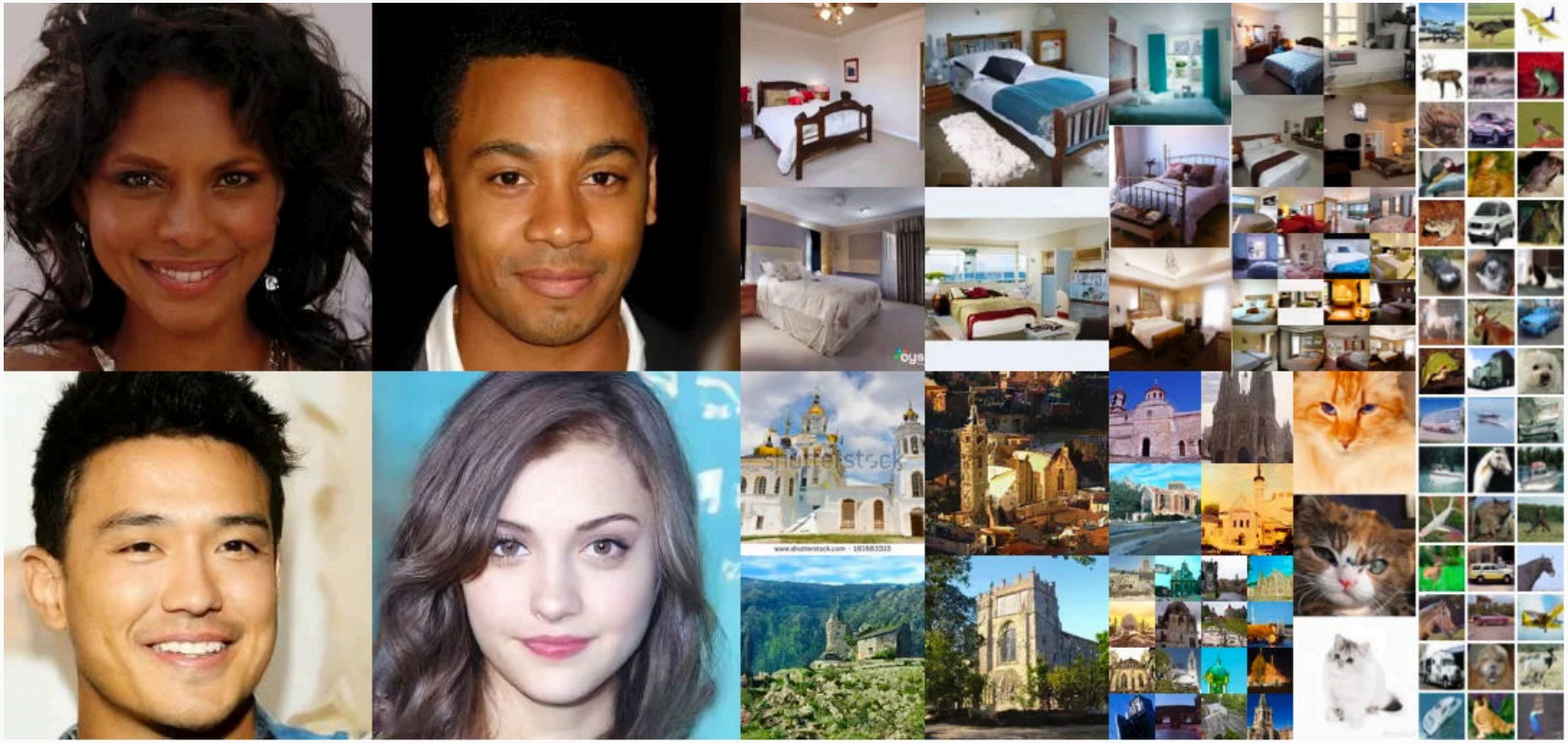
- But then DDPM makes the following observations:
  - the variances of forward diffusion β_t are constant hyperparameters, they are not trained, so all of the q's are not trained; since $p_\theta(\mathbf{x}_T)$ is a fixed distribution from which we will sample, L_T is a constant;
  - for the intermediate steps the variances in $p_\theta(\mathbf{x}_t \mid \mathbf{x}_{t+1})$ are also not trained, and one can find a simple closed form for L_t;
  - most importantly, let us introduce a separate discrete decoder for L_0; if the data consists of integers from 0 to 255, rescaled to $[-1, 1]$ (this is the natural representation for images), DDPM models

$$p_\theta(\mathbf{x}_0 \mid \mathbf{x}_1) = \prod_{i=1}^{D} \int_{\delta_-(x_0,i)}^{\delta_+(x_0,i)} N\left(x \middle| \mu_{\theta,i}(\mathbf{x}_1), \sigma_1^2\right) \mathrm{d}x,$$

    where i runs over the pixels, $\mu_\theta(\mathbf{x}_1)$ is the decoder, and the integration limits are $\frac{1}{255}$ on each side of $x_{0,i}$
- That is, one can substitute a different model at the last step and use $\mu_\theta(\mathbf{x}_1)$ without noise during sampling

DDPM

- DDPM already gave excellent generation, comparable to the best GANs of that time:

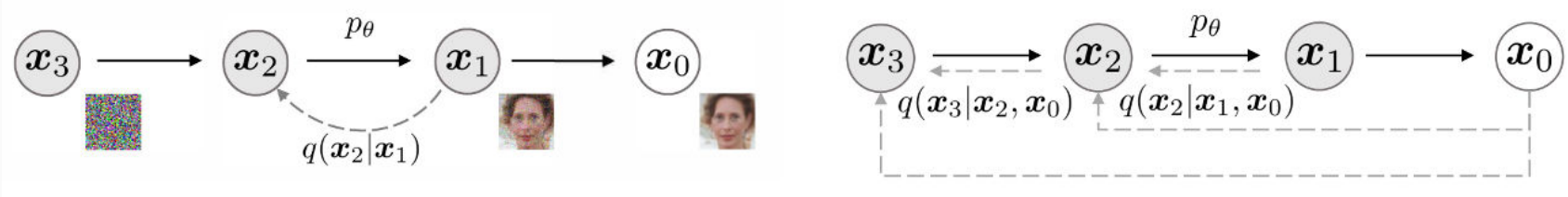


DDIM

- The next step came very soon: “Denoising Diffusion Implicit Models” (DDIM; Song et al., 2020)
- An important drawback of all diffusion models so far is that they are *very slow*
- Generation has to go through all the diffusion steps, and there are literally thousands of them, and they run sequentially, they do not parallelize
- Song et al. mention that sampling from a trained GAN is about 1000 times faster than sampling from DDPM for the same image size

DDIM

- How can we speed up diffusion models?
- Song et al. generalize diffusion models and in particular DDPM
- The loss function L does not depend directly on the joint distribution $q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)$, only on the marginals $q(\mathbf{x}_t \mid \mathbf{x}_0)$
- This means that we can reuse the same loss function for a different joint distribution, as long as it has the same marginals
- In particular, we can even generalize to non-Markovian diffusion processes, provided that we can find a reverse Markov chain for them:



DDIM

- For DDIM we will define the diffusion process in terms of its reverse form:

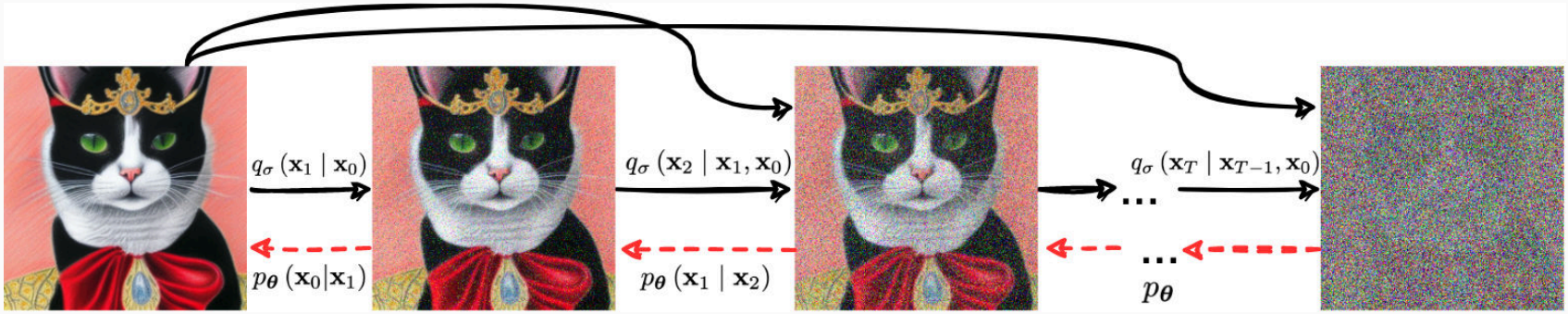
$$q_\sigma\left(\mathbf{x}_{1:T} \mid \mathbf{x}_0\right) = q_\sigma\left(\mathbf{x}_T \mid \mathbf{x}_0\right) \prod_{t=2}^{T} q_\sigma\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0\right).$$

- Now we can express the forward diffusion distributions by Bayes' theorem:

$$q_\sigma\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{x}_0\right) = \frac{q_\sigma\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0\right) q_\sigma\left(\mathbf{x}_t \mid \mathbf{x}_0\right)}{q_\sigma\left(\mathbf{x}_{t-1} \mid \mathbf{x}_0\right)}.$$

DDIM

- And now we can show that the resulting process has exactly the same marginals, so the reverse diffusion can be trained with the same loss function, and it will be a Markov chain:

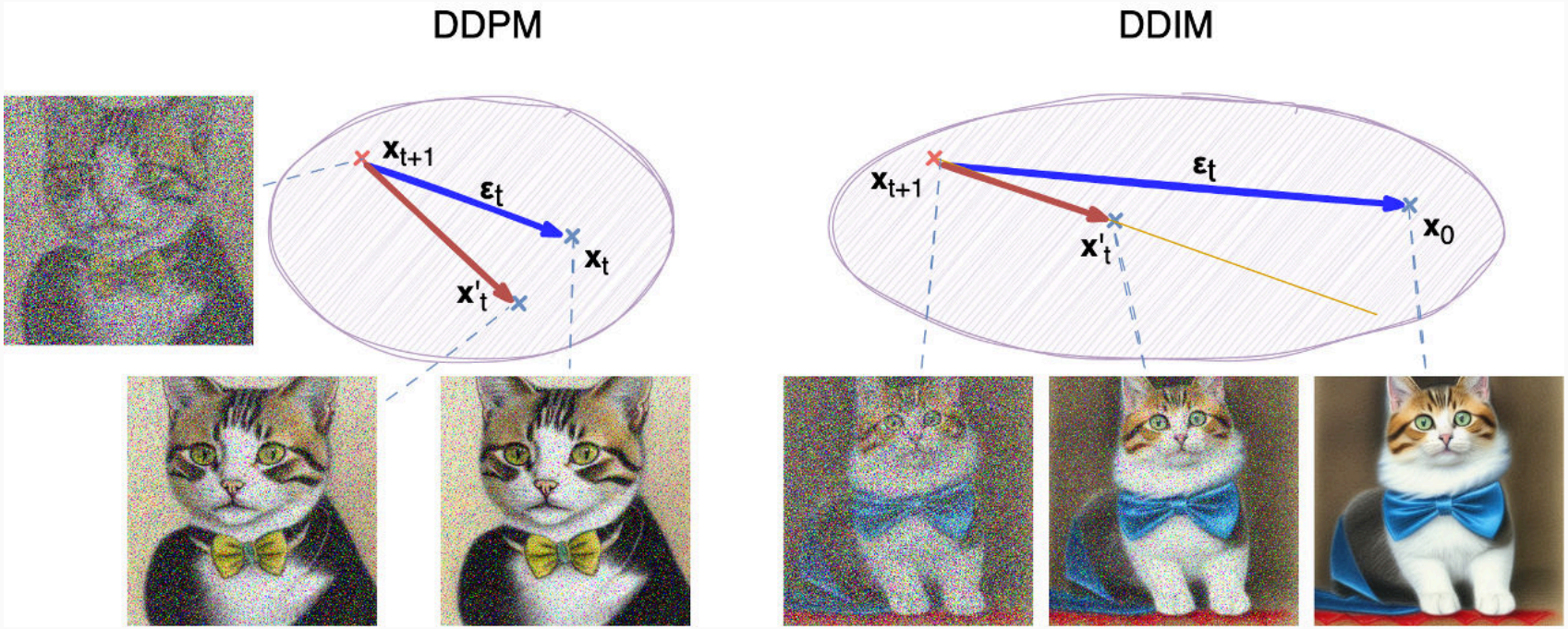


DDIM

- So far this does not help much: we have extended the class of distributions for forward diffusion, but sampling a new image still has to go through all the steps of reverse diffusion
- But now, instead of approximating the random noise ϵ_t that leads from $\mathbf{x}_t$ to $\mathbf{x}_{t+1}$, we can approximate the random noise ϵ_t that will lead directly from $\mathbf{x}_0$ to $\mathbf{x}_{t+1}$!
- And when we move in the reverse direction, we approximate the direction not toward $\mathbf{x}_t$, but directly toward $\mathbf{x}_0$, and we can move in that direction faster

DDIM

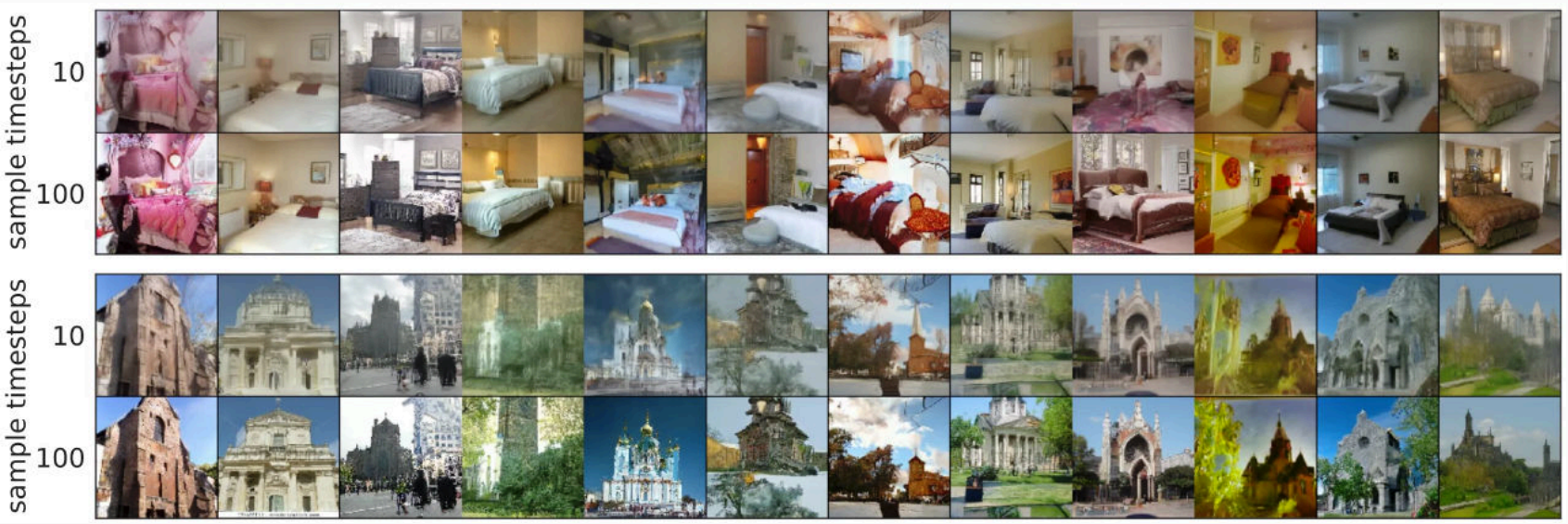
- An illustration:



- It is hard to say which approximation is better, but now we have separated the dependence of ϵ_t on $\mathbf{x}_0$, and we can jump several steps at once, moving from $\mathbf{x}_t$ to $\mathbf{x}_{t+k}$ in one step with an increased ϵ!

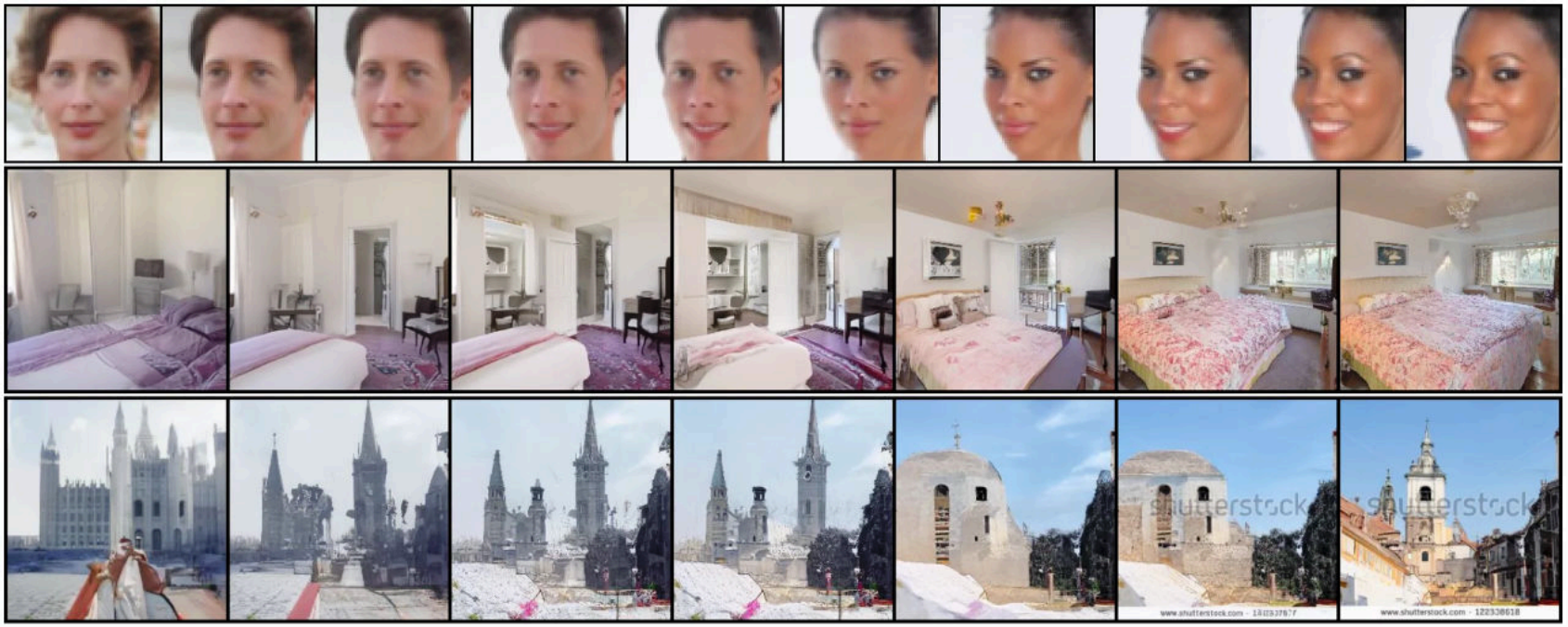
DDIM

- This led to a 10x–100x speedup compared to DDPM without loss of quality:



DDIM

- Generation in DDIMs does not have to be stochastic either; one can set the variance of reverse diffusion to zero during generation
- And now the latent code in the space $\mathbf{x}_T$ corresponds to exactly one image, and DDIM is a good model for latent representations; here, for example, are interpolations in the latent space:

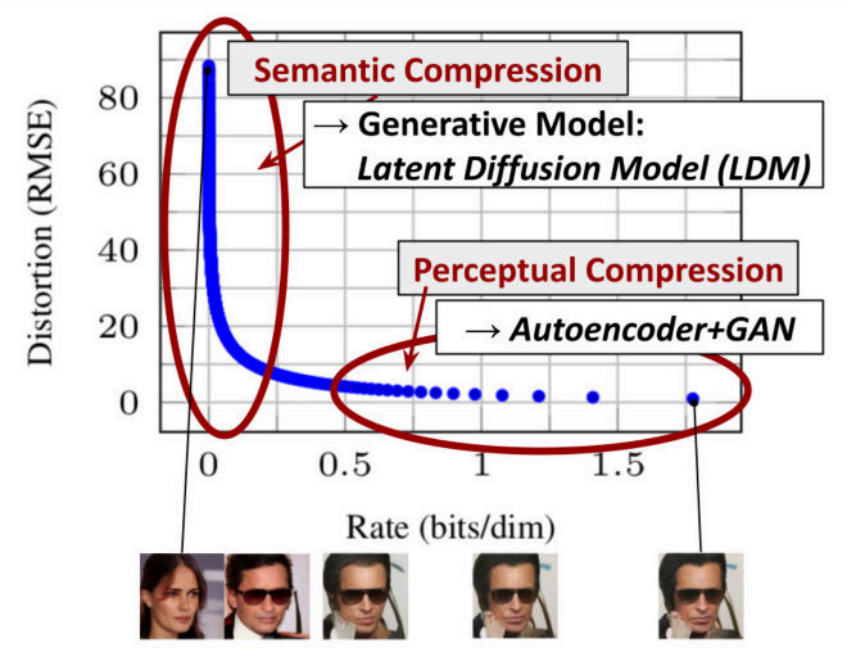


- DDPM could not interpolate, because a great deal of noise is

Stable Diffusion

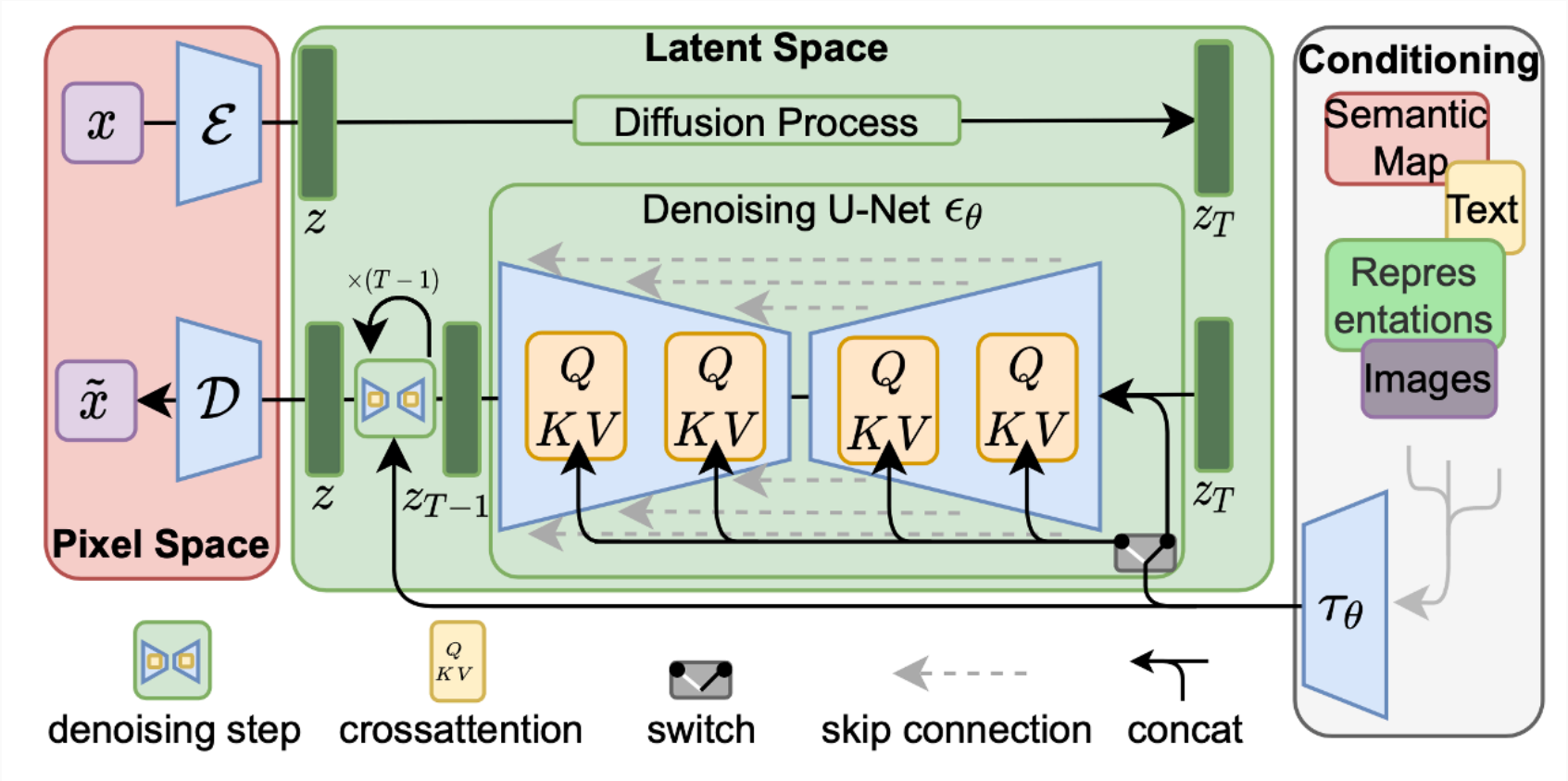
Stable Diffusion

- Another breakthrough came when this diffusion process started to be used *in latent space*
- Stable Diffusion (Rombach, Blattmann et al., 2022):



Stable Diffusion

- Stable Diffusion (Rombach, Blattmann et al., 2022):

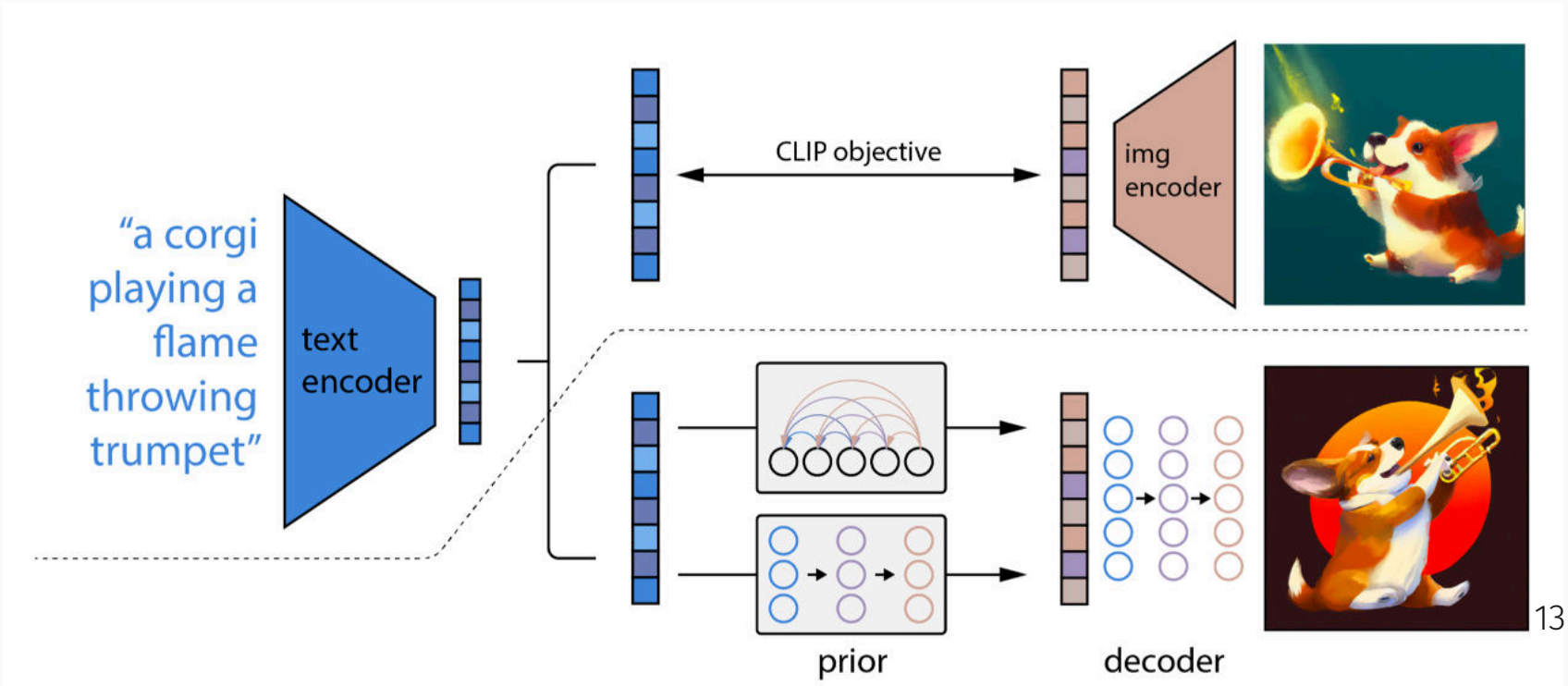


Stable Diffusion

- unCLIP, also known as DALL-E 2 (Ramesh et al., 2022), trains

$$p(\mathbf{x}|y) = p(\mathbf{x}|\mathbf{c}, y)\, p(\mathbf{c}|y),$$

where the prior model $p(\mathbf{c}|y)$ builds a CLIP embedding from text, and in $p(\mathbf{x}|\mathbf{c}, y)$ first a "noise" (prior) for the image is generated, and then a diffusion model draws the image



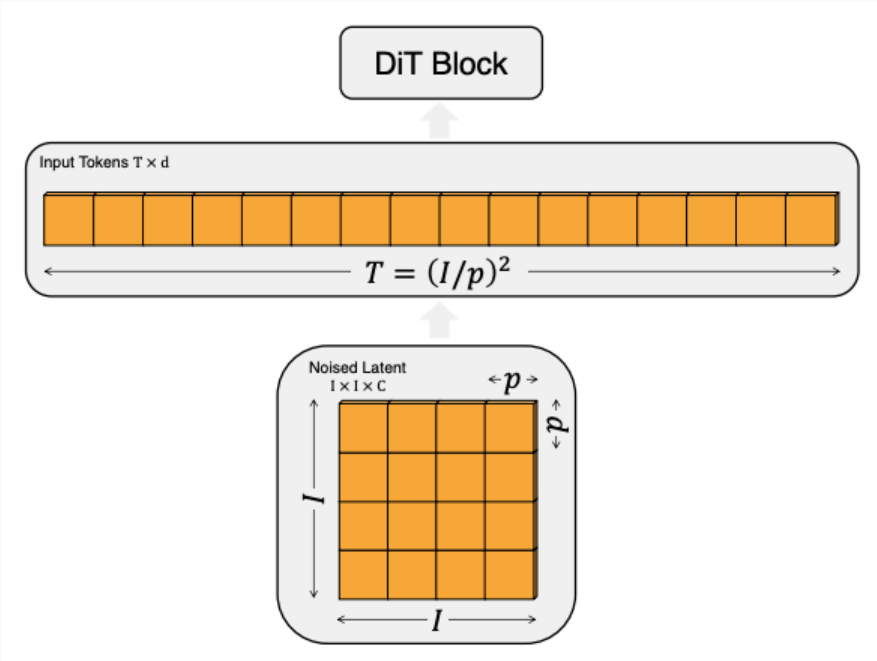
DALL-E 2

Midjourney

Diffusion Transformers

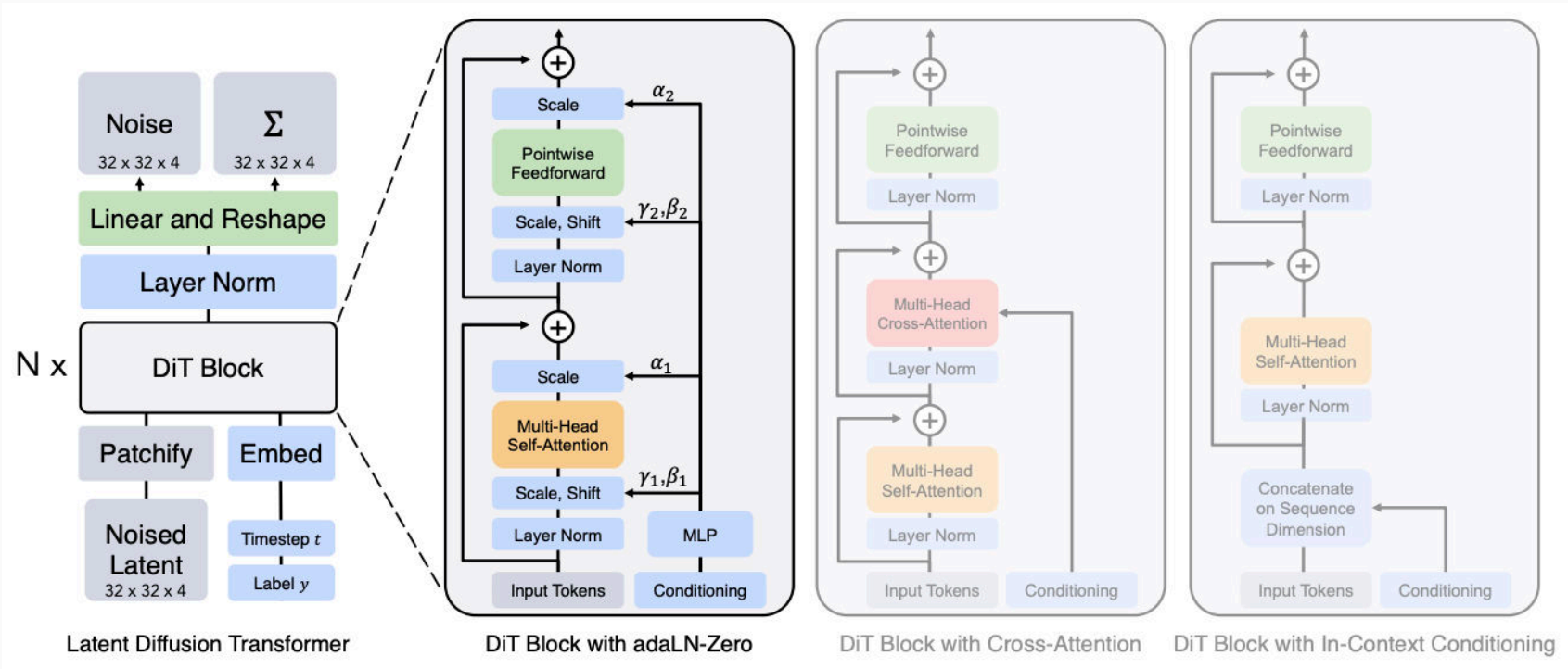
Diffusion Transformers

- A fully modern architecture by now – DiT (Peebles, Xie, 2022)
- This is a latent diffusion model, but instead of a U-Net-like architecture for the latent codes it uses a transformer
- The transformer input is an $I \times I \times C$ tensor, "patchified" into a sequence of $p \times p$ patches of length $T = (I/p)^2$; p is a hyperparameter that affects complexity quadratically



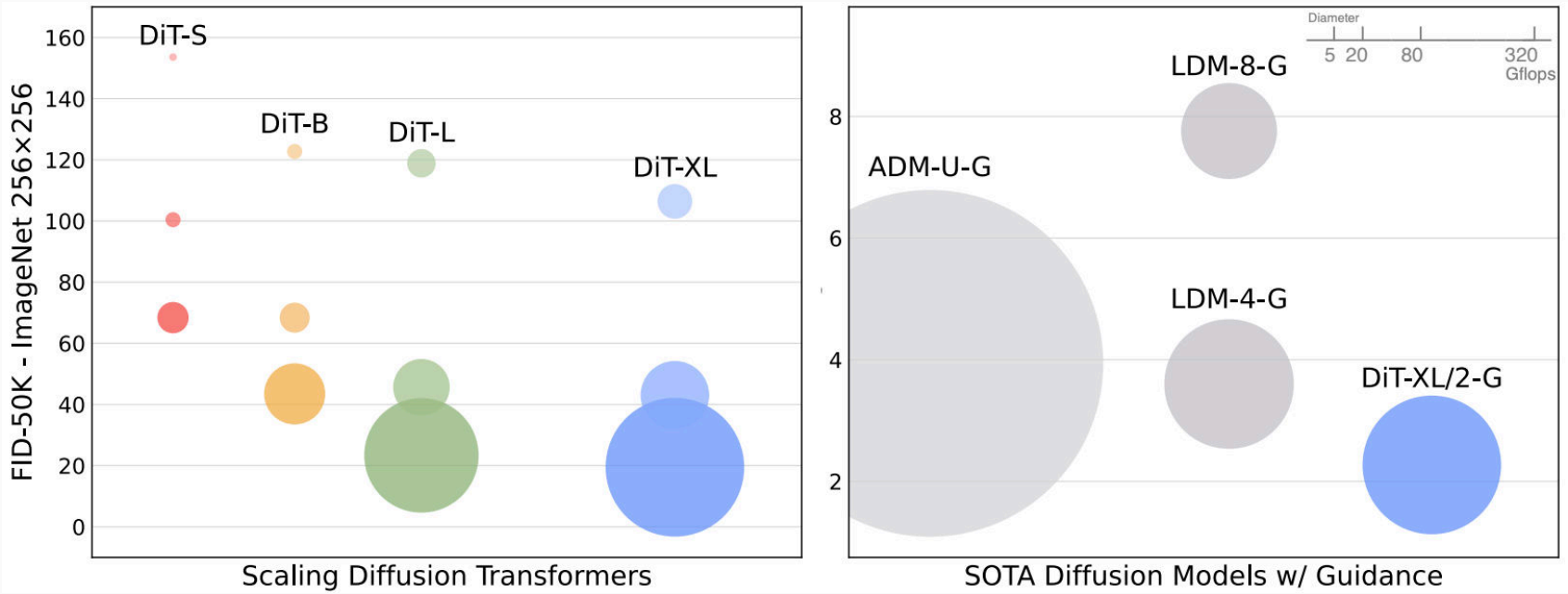
Diffusion Transformers

- Then it is processed by DiT blocks; the interesting part here is adaptive layer norm with parameters taken from the condition



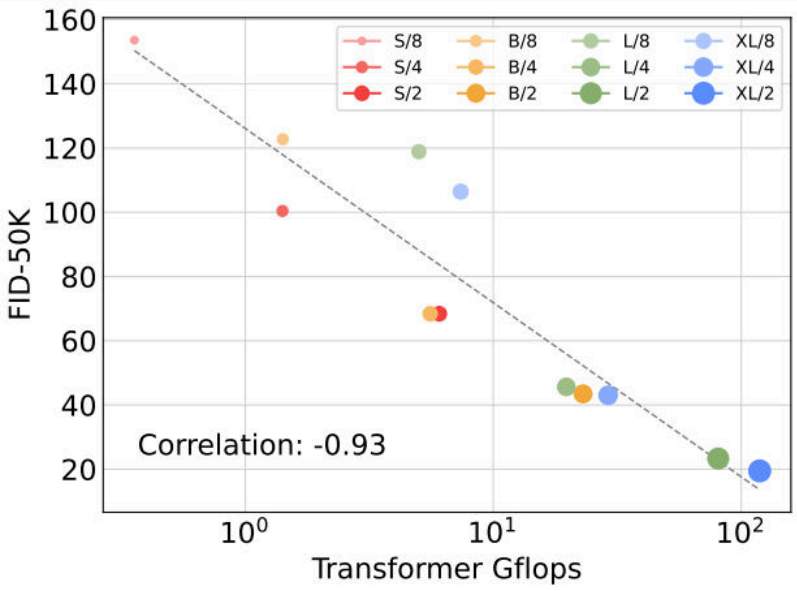
Diffusion Transformers

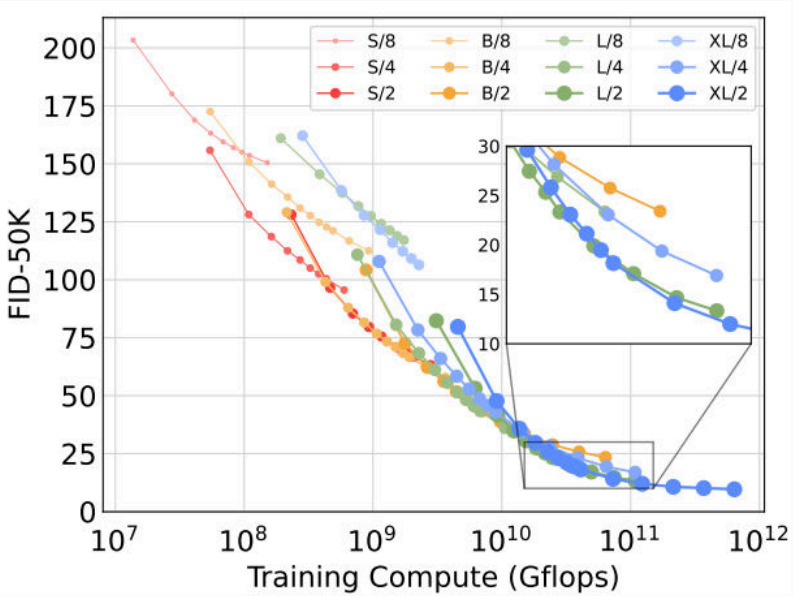
- It turns out that DiTs are much more computationally efficient:



Diffusion Transformers

- Transformer GFlops are strongly correlated with quality, and large DiT models use computation more efficiently:





Diffusion Transformers

- Good samples:

Flux

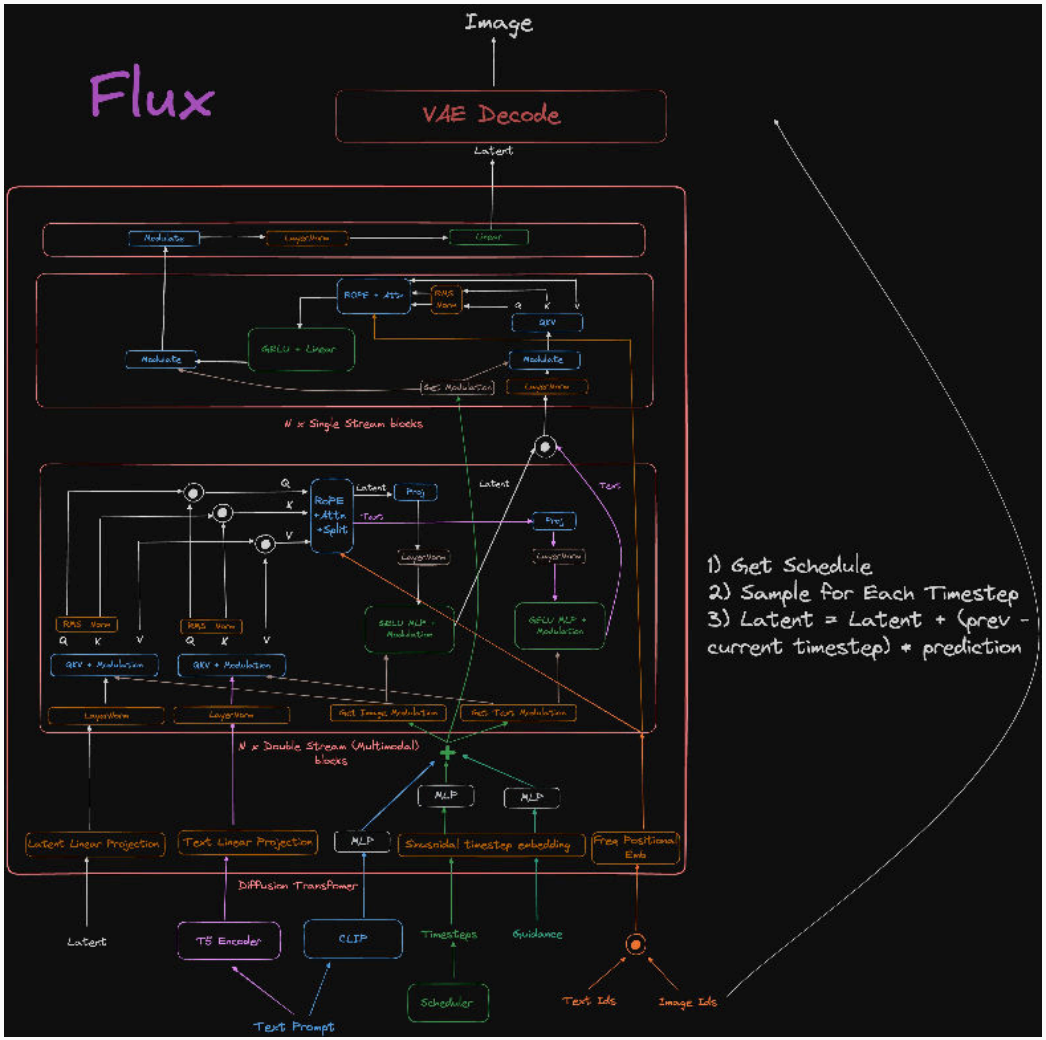
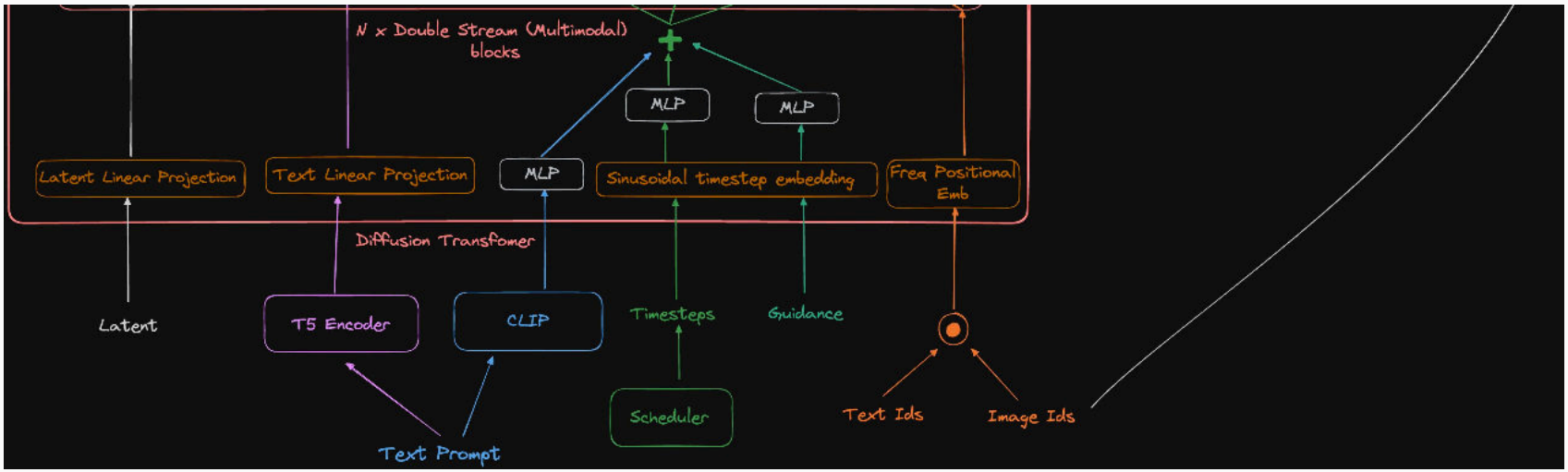
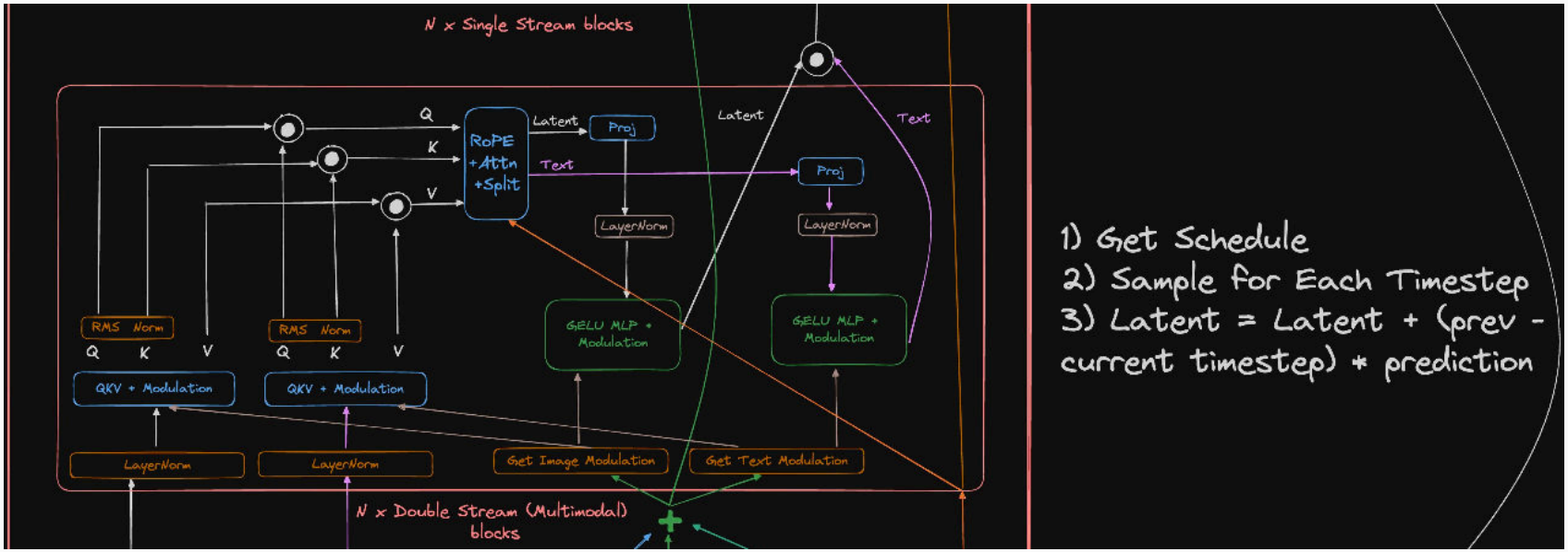


Image
Flux
VAE Decode
Latent
Modulate
LayerNorm
Linear
RoPE + Attn
RMS Norm
Q
K
V
QKV
GELU + Linear
Modulate
LayerNorm
N x Single Stream blocks
RoPE
+Attn
+Split
Latent
Proj
Text
LayerNorm
RMS Norm
QKV + Modulation
GELU MLP + Modulation
N x Double Stream (Multimodal) blocks
MLP
Latent Linear Projection
Text Linear Projection
Sinusoidal timestep embedding
Freq Positional Emb
Diffusion Transformer
T5 Encoder
CLIP
Timesteps
Guidance
Scheduler
Text Ids
Image Ids
Text Prompt
1) Get Schedule
2) Sample for Each Timestep
3) Latent = Latent + (prev - current timestep) * prediction

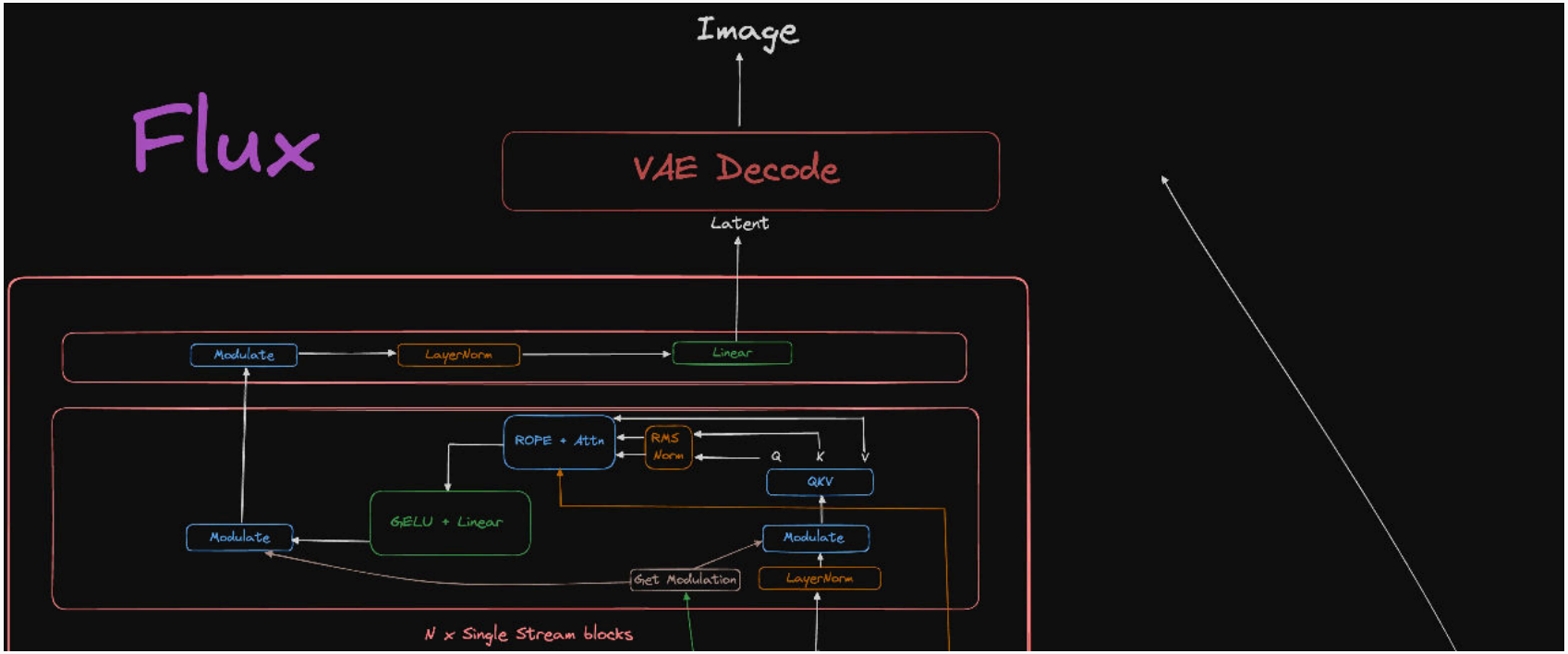
Flux


N x Double Stream (Multimodal) blocks
MLP
MLP
Latent Linear Projection
Text Linear Projection
MLP
Sinusoidal timestep embedding
Freq Positional Emb
Diffusion Transfomer
Latent
T5 Encoder
CLIP
Timesteps
Guidance
Scheduler
Text Ids
Image Ids
Text Prompt

Flux


N x Single Stream blocks
Q
K
V
RoPE
+Attn
+Split
Latent
Proj
Text
Proj
LayerNorm
LayerNorm
Latent
Text
RMS Norm
Q K V
RMS Norm
Q K V
GELU MLP + Modulation
GELU MLP + Modulation
QKV + Modulation
QKV + Modulation
LayerNorm
LayerNorm
Get Image Modulation
Get Text Modulation
N x Double Stream (Multimodal) blocks
1) Get Schedule
2) Sample for Each Timestep
3) Latent = Latent + (prev - current timestep) * prediction

Flux


Flux
Image
VAE Decode
Latent
Modulate
LayerNorm
Linear
ROPE + Attn
RMS Norm
Q
K
V
QKV
GELU + Linear
Modulate
Modulate
Get Modulation
LayerNorm
N x Single Stream blocks

Flux

Flux

Flux

Flux

Flux

What's next?

Recent news in diffusion models

- What happened after DiT in diffusion models?
- DPM-Solver++ (Lu et al., 2022): how to speed up DDIM further?
- As we saw above, DDIM is a first-order step for integrating a certain ODE
- But there is a whole science of how to solve ODEs better!
- Let us pick some multistep second- or third-order scheme; it will reduce the local error, and in the end we will be able to take even wider steps, and even fewer steps will be needed

Recent news in diffusion models

- For example, DPM-Solver++ is indeed more stable and works better than other variants (which we will not discuss):

DDIM (order = 1) (Song et al., 2021a)

PNDM (order = 2) (Liu et al., 2022b)

DEIS-1 (order = 2) (Zhang & Chen, 2022)

DEIS-2 (order = 3) (Zhang & Chen, 2022)

DPM-Solver (order = 2) (Lu et al., 2022)

DPM-Solver-3 (order = 3) (Lu et al., 2022)

†DDIM (thresholding) (Saharia et al., 2022b)

DPM-Solver++ (order = 2) (**ours**)

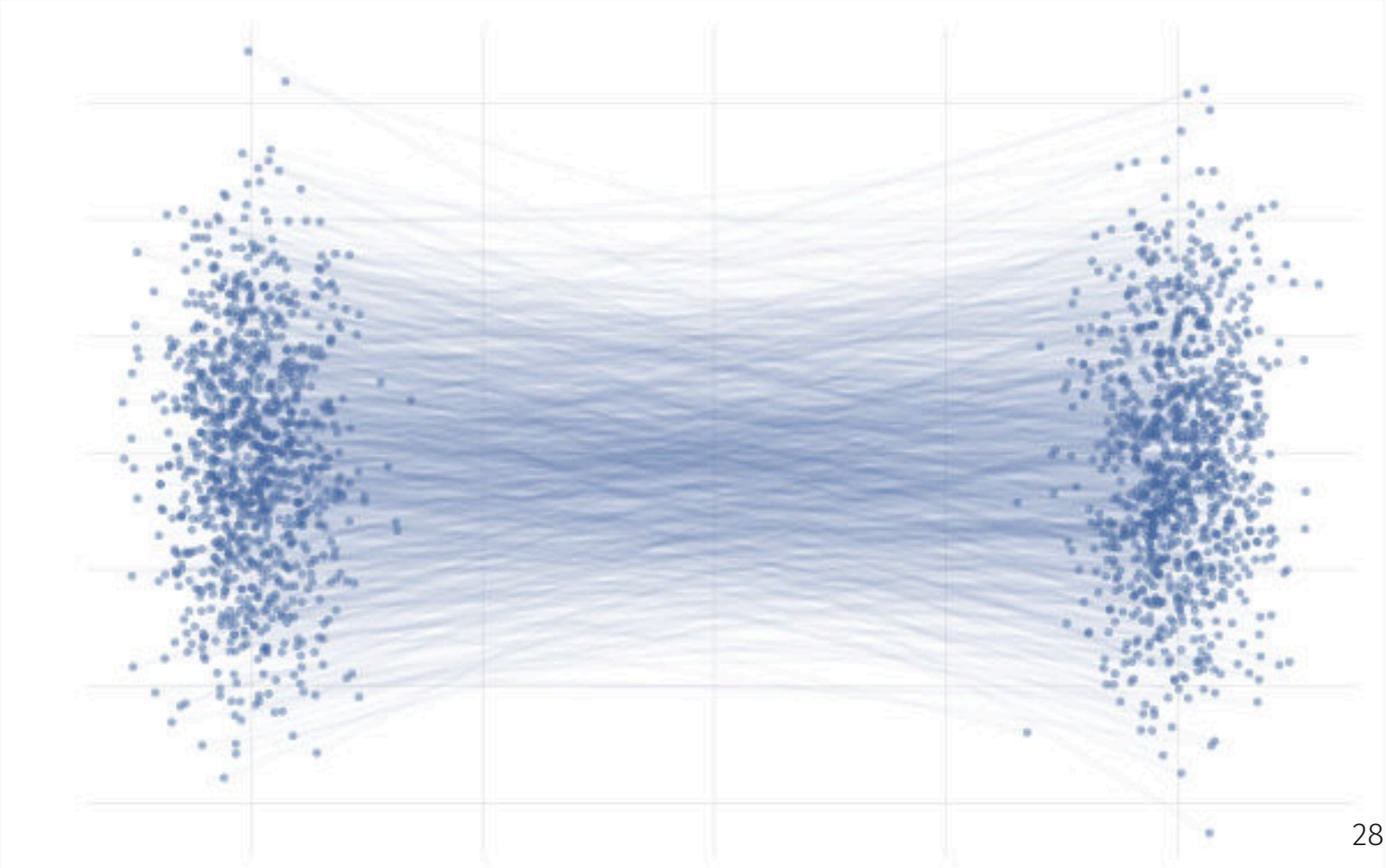
Recent news in diffusion models

- UniPC (Zhao et al., 2023): a “predictor-corrector”-type method
- They derive a universal correcting step (UniC) that can be added after any base predictor to increase the order of accuracy without additional model calls, and they derive the matching UniP predictor

DPM-Solver++ [26]	UniPC (**Ours**)	DPM-Solver++ [26]	UniPC (**Ours**)
			
“black and white cat is sitting on top of a wooden bench”		*“vase filled with flowers sitting on a table”*	

Recent news in diffusion models

- And the most interesting further development is, of course, flow matching:



Recent news in diffusion models

- Many modern generative models are based on it:

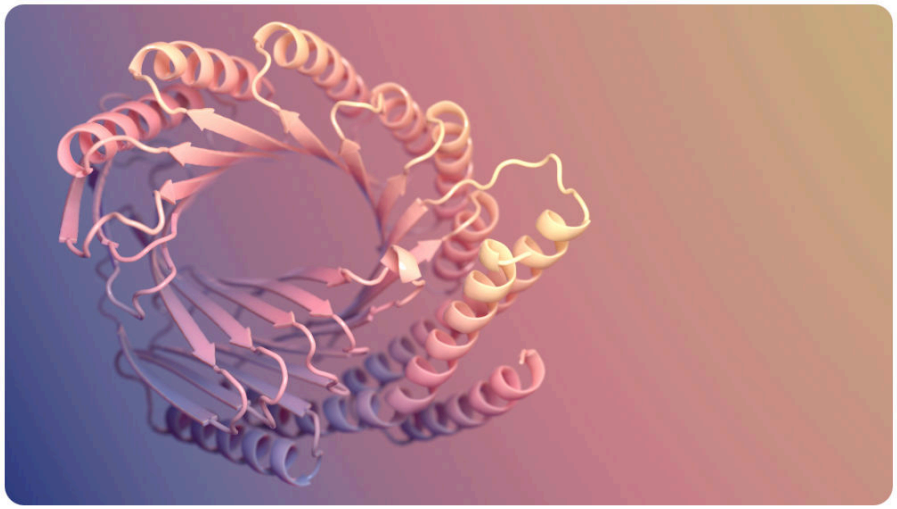
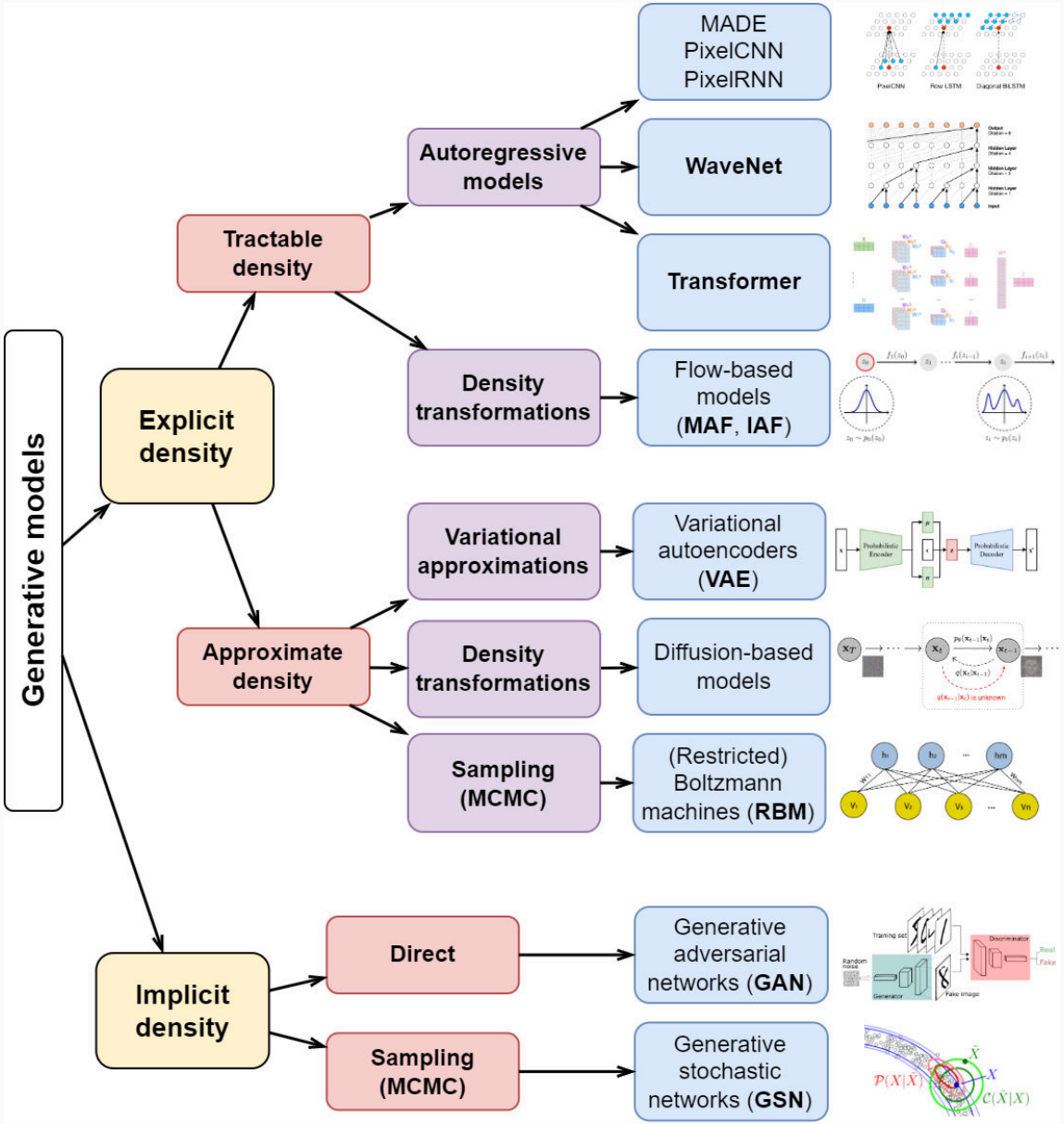


Figure 1: Protein generated by RFDiffusion (Watson et al., 2023).

Figure 2: Image from DALL-E 3 (Betker et al., 2023).

- But that's a different story...

Deep generative models



Thank you!

Thank you for your attention!