CS60010: Deep Learning

Spring 2023

Sudeshna Sarkar

Diffusion Models for Generation

12 April 2023

Diffusion models have become state-of-the-art for generative modeling

Abstract painting of an artificial intelligent agent

A hedgehog using a calculator.

A corgi wearing a red bowtie and a purple party hat.

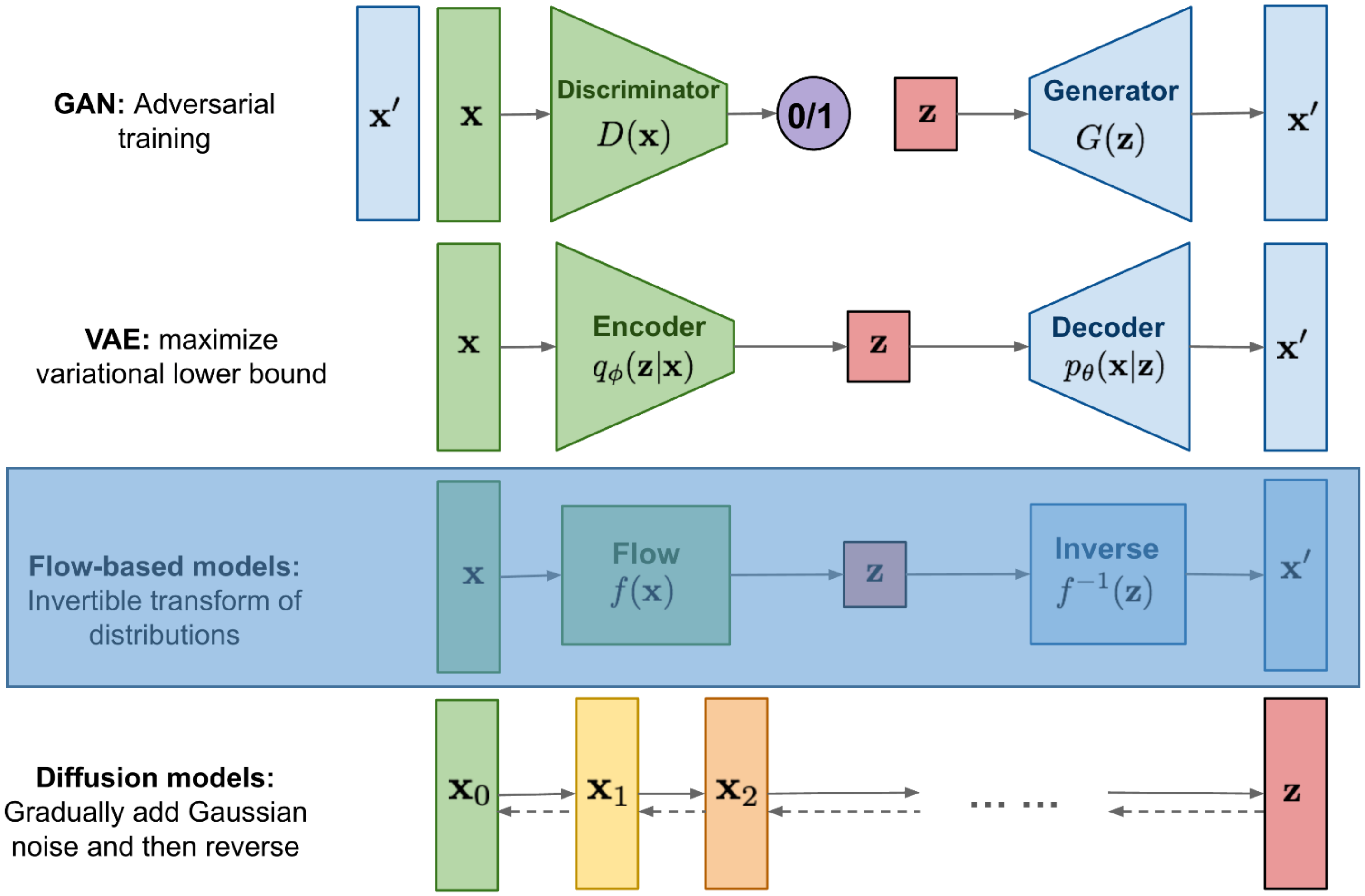
A transparent sculpture- of a duck made out of glass.

A photo of a Corgi dog riding a bike in Times Square. It is wearing sunglasses and a beach hat.

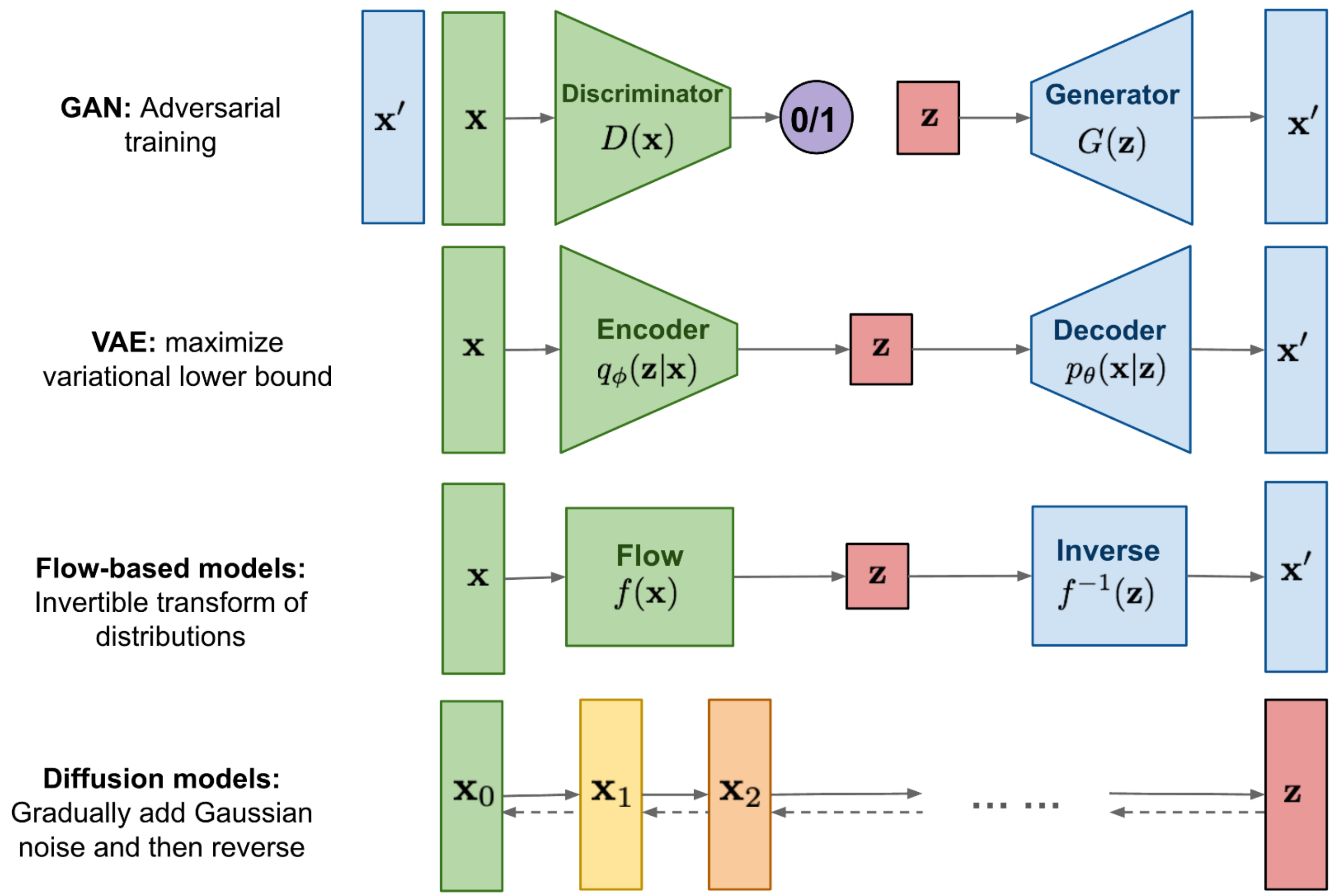
Pomeranian king with tiger soldiers.

Zebras roaming in the field.

Generative Models



Generative Models

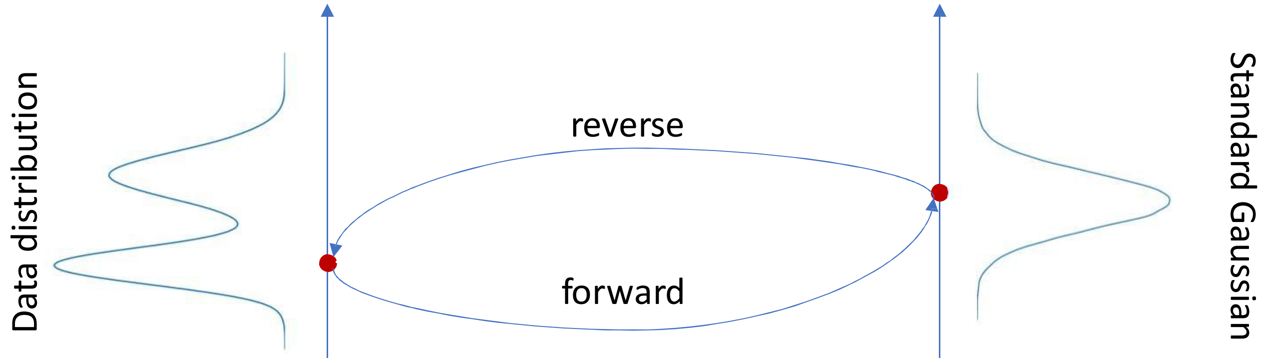


- GAN models are known for potentially unstable training and less diversity in generation due to their adversarial training nature.
- VAE relies on a surrogate loss.
- Flow models have to use specialized architectures to construct reversible transform.

Diffusion models are inspired by non-equilibrium thermodynamics. They define a Markov chain of diffusion steps to slowly add random noise to data and then learn to reverse the diffusion process to construct desired data samples from the noise.

Diffusion Models: High level overview

- Diffusion models are probabilistic models used for image generation
- They involve reversing the process of gradually degrading the data
- Consist of two processes:
  - The forward process: data is progressively destroyed by adding noise across multiple time steps
  - The reverse process: using a neural network, noise is sequentially removed to obtain the original data

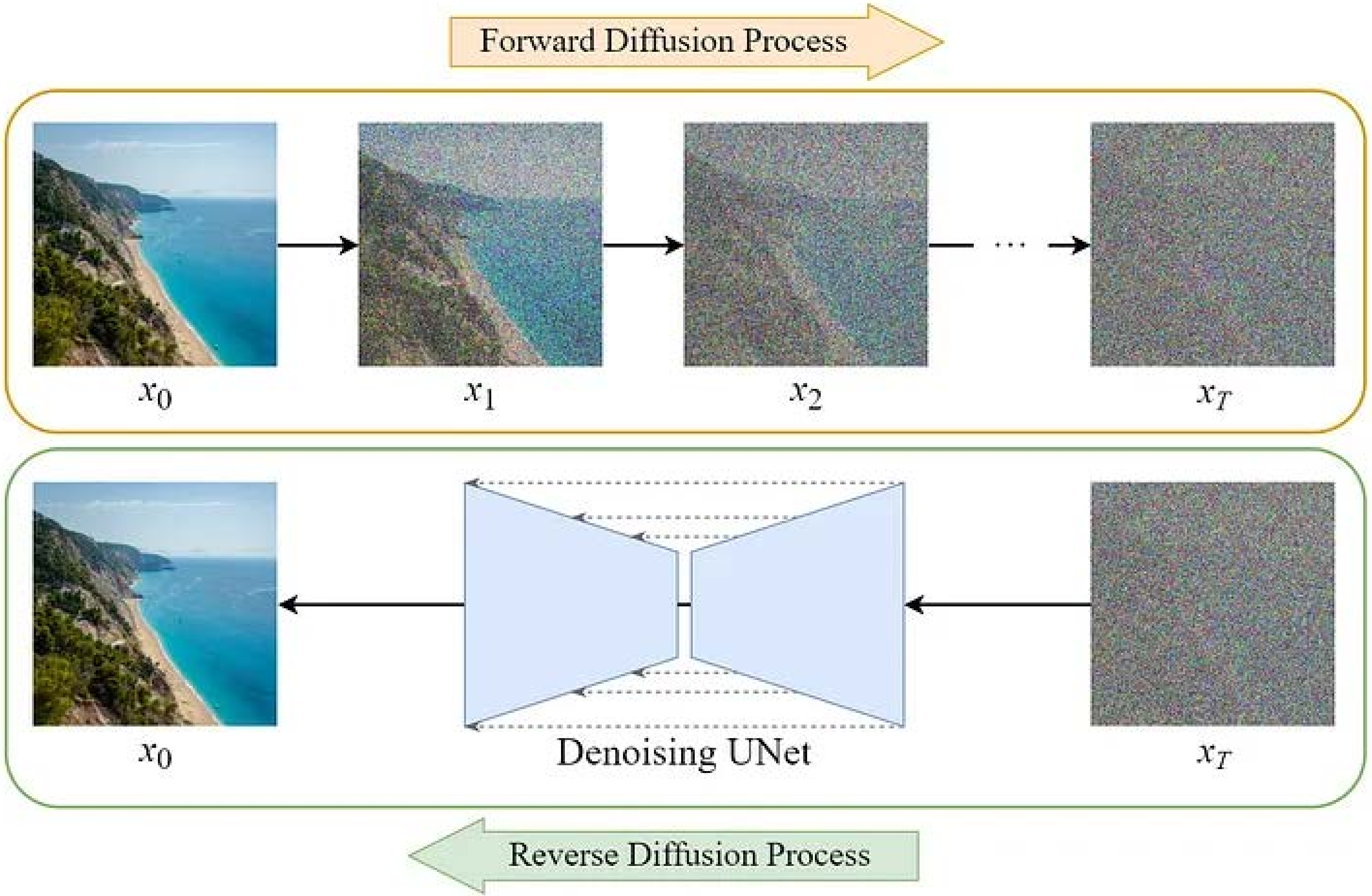


High-level overview

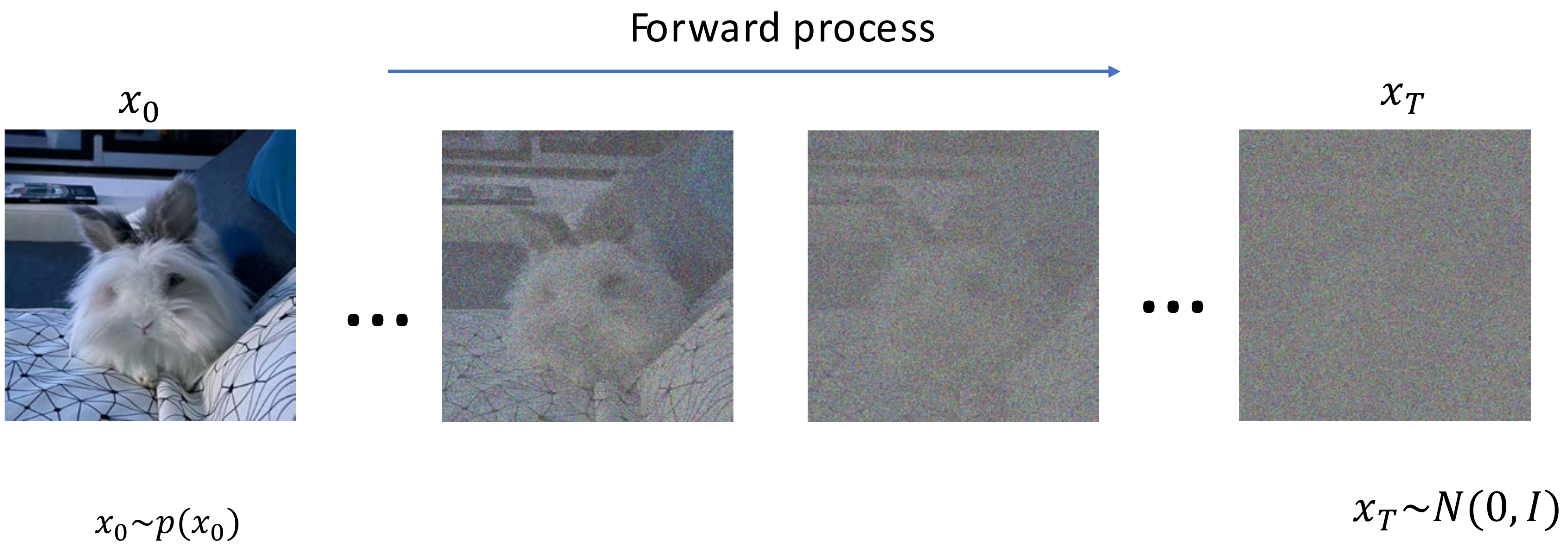
- Three categories:
    1. **Denoising Diffusion Probabilistic Models (DDPM)**
    2. Noise Conditioned Score Networks (NCSN)
    3. Stochastic Differential Equations (SDE)

1. Forward Diffusion Process → add noise to the image.

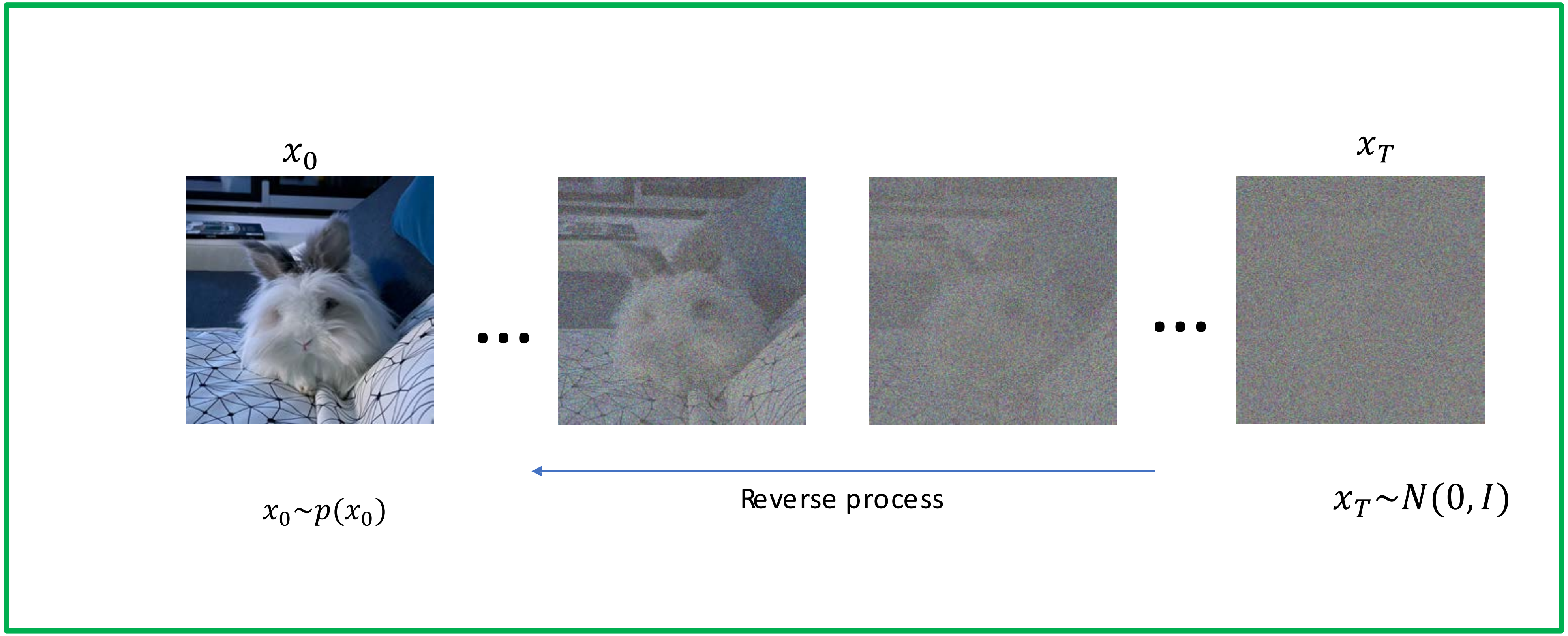
2. Reverse Diffusion Process → remove noise from the image.



Denoising Diffusion Probabilistic Models (DDPMs)



Denoising Diffusion Probabilistic Models (DDPMs)

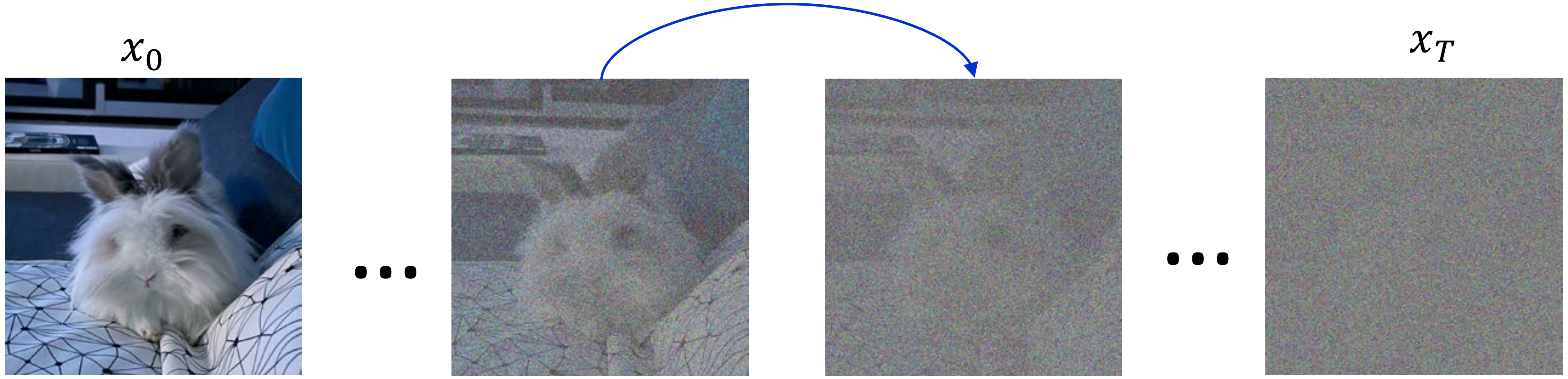


DDPM Forward process (Iterative)

$$x_t \sim p(x_t|x_{t-1}) = \mathcal{N}\left(x_t; \sqrt{1-\beta_t} \cdot x_{t-1}, \beta_t I\right) \qquad \beta_t \ll 1$$

The image is replaced with noise

x_0 $\qquad$ x_T



$x_0 \sim p(x_0)$ $\qquad$ x_{t-1} $\qquad$ x_t $\qquad$ $x_T \sim N(0, I)$

Forward Diffusion

- Given a data point sampled from a real data distribution $x_0 \sim p(x)$
- *forward diffusion process*: add small amount of Gaussian noise to the sample in **T** steps
  - producing a sequence of noisy samples $x_1, x_2, \dots, x_T$.
- The step sizes are controlled by a *variance schedule* $\{\beta_t \in (0,1)\}_{t=1}^{T}$.

$$p(x_t|x_{t-1}) = \mathcal{N}\left(x_t; \sqrt{1-\beta_t} \cdot x_{t-1}, \beta_t I\right) \quad p(x_{1:T}|x_0) = \prod_{t=1}^{T} p(x_t|x_{t-1})$$

The data sample x_0 gradually loses its distinguishable features.

Eventually when $T \to \infty$, x_T is equivalent to an isotropic Gaussian distribution.

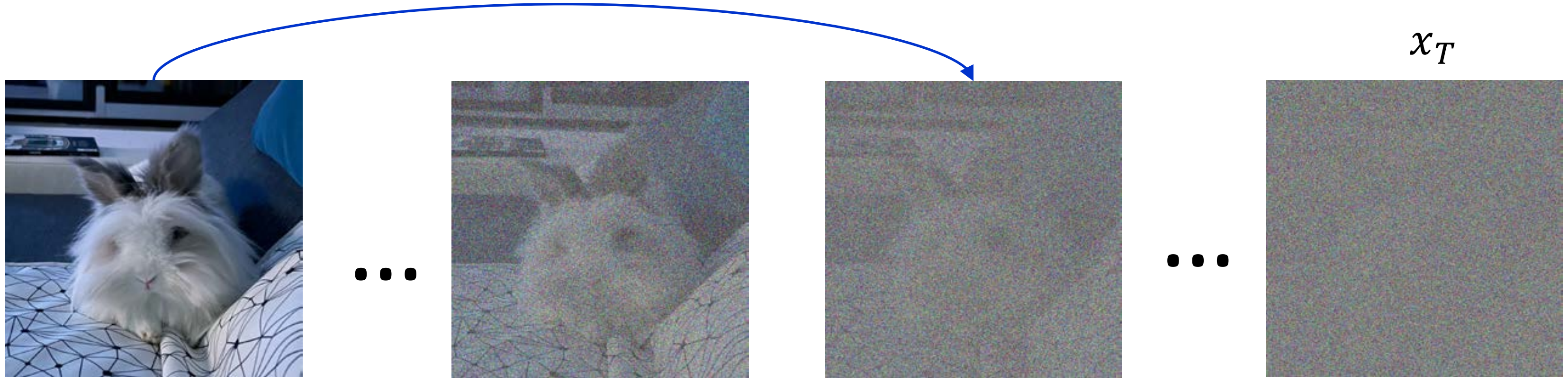
DDPM Forward process. Ancestral sampling (One Shot)

$$x_t \sim p(x_t|x_0) = \mathcal{N}\left(x_t; \sqrt{\widehat{\beta}_t} \cdot x_0, (1 - \widehat{\beta}_t)I\right)$$

$$\widehat{\beta}_t = \prod_{i=1}^{t} \alpha_i$$

$$\alpha_t = 1 - \beta_t$$

x_T



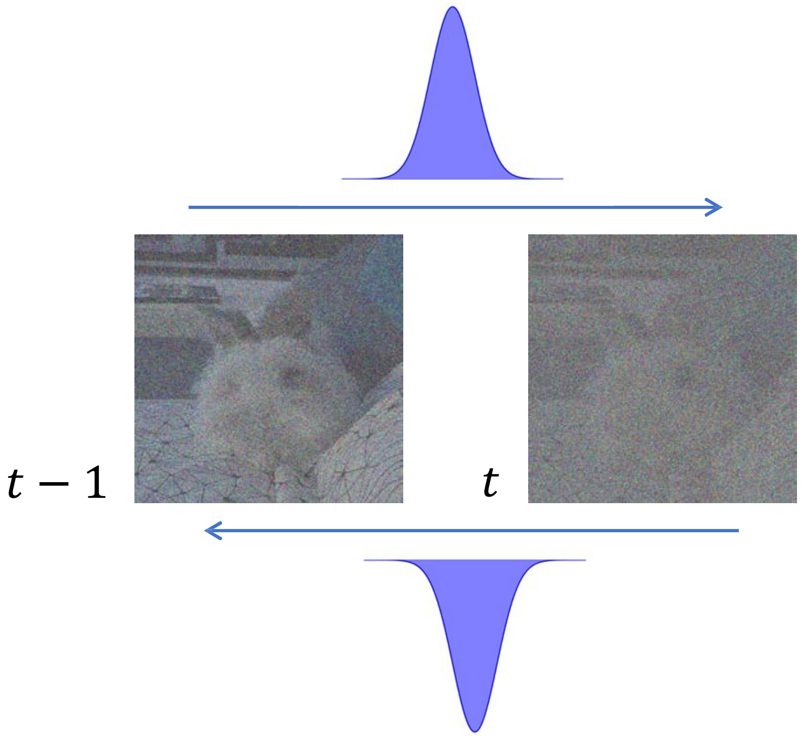
$x_0 \sim p(x_0)$ x_{t-1} x_t $x_T \sim N(0, I)$

DDPMs. Properties of β_t

1. $\beta_t \ll 1, t = \overline{1,T}$

$$x_t \sim p(x_t|x_{t-1}) = \mathcal{N}\left(x_t;\ \sqrt{1-\beta_t}\cdot x_{t-1}, \beta_t I\right)$$

x_t is created with a small step modeled by β_t

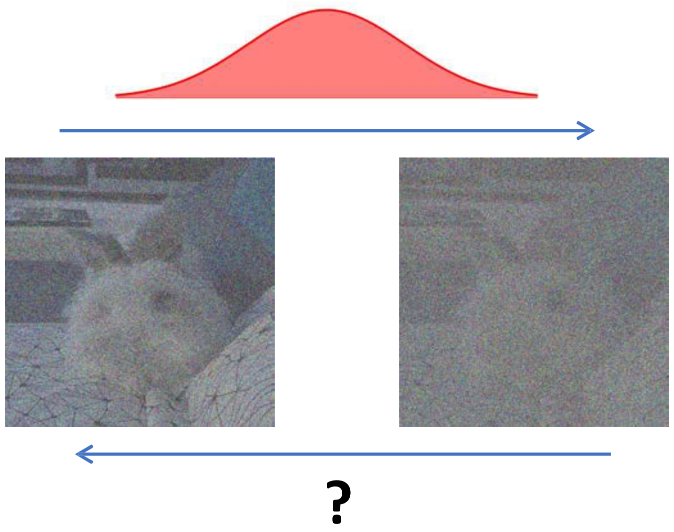


x_{t-1} comes from a region close to x_t
Therefore we can model with Gaussian

$$x_{t-1} \sim p(x_{t-1}|x_t) = \mathcal{N}(x_{t-1};\ \mu(x_t,t), \Sigma(x_t,t))$$

DDPMs. Properties of β_t

X $\beta_t \ll 1, t = \overline{1, T}$



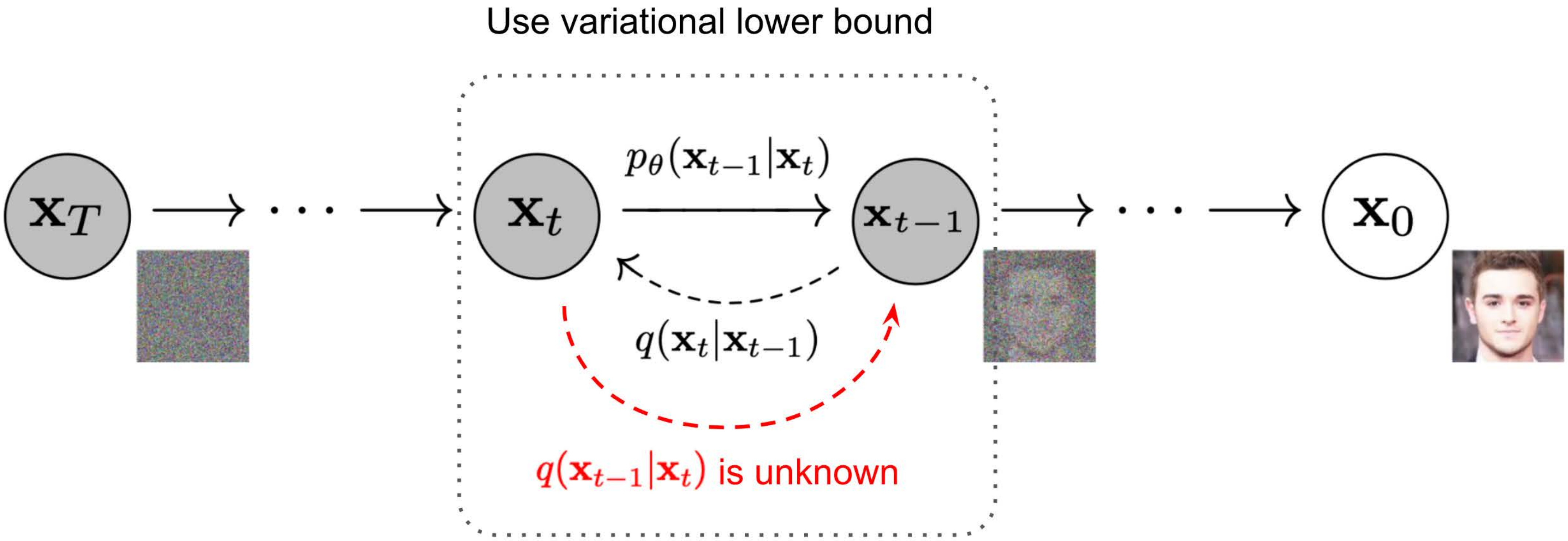
Less certain where was the x_{t-1},
because we could have reached x_t
from many more regions.

DDPMs. Properties of β_t

1. $\beta_t \ll 1, t = \overline{1, T}$, x_T is pure noise
2. T is large

x_0

x_T



The Markov chain of forward (reverse) diffusion process of generating a sample by slowly adding (removing) noise.

Reverse diffusion process

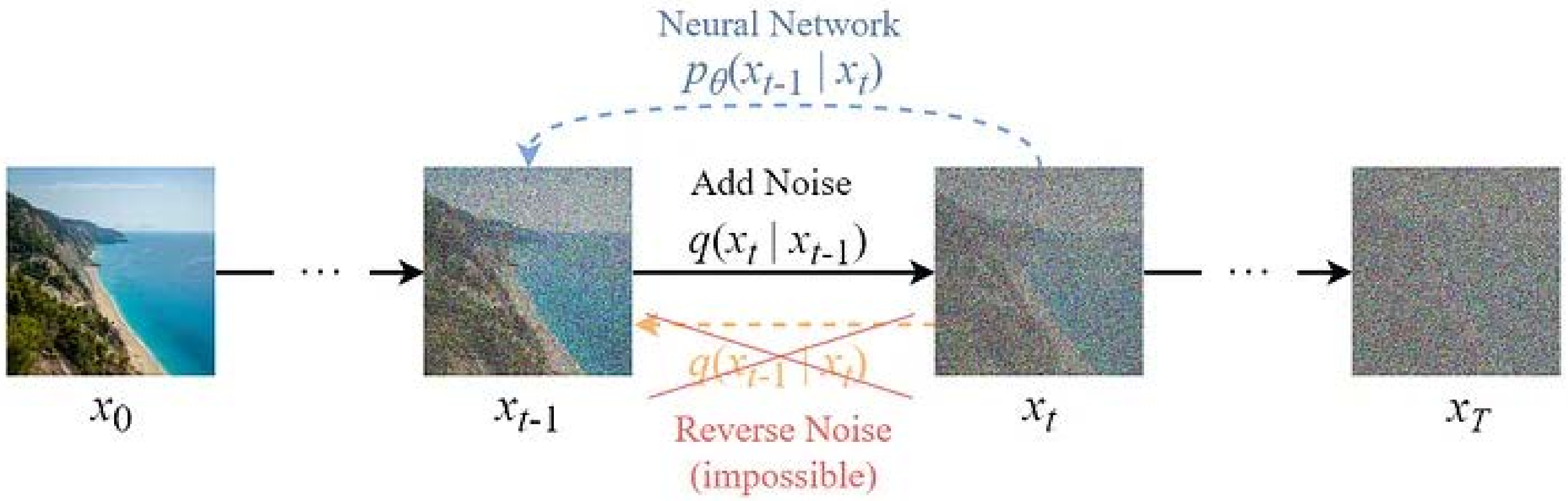
- f we can reverse the above process and sample from $q(x_{t-1}|x_t)$ we will be able to recreate the true sample from a Gaussian noise input $x_T \sim \mathcal{N}(0,1)$.
- we cannot easily estimate $p(x_{t-1}|x_t)$ because it needs to use the entire dataset and therefore we need to learn a model p_θ to approximate these conditional probabilities in order to run the *reverse diffusion process*.

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

- The reverse conditional probability is tractable when conditioned on x_0

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

Reverse Diffusion Process



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

Approximated Distribution $\quad p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1};\ \mu_\theta(x_t, t),\ \Sigma_\theta(x_t, t))$

Learnable parameters
(Neural Network)

Unlike the forward process, we cannot use $\boldsymbol{q(x_{t-1}|x_t)}$ to reverse the noise since it is intractable (uncomputable).

- Unlike the forward process, we cannot use $p(x_{t-1}|x_t)$ to reverse the noise since it is intractable (uncomputable).
- Thus we need to train a neural network $p\theta(x_{t-1}|x_t)$ to approximate $p(x_{t-1}|x_t)$.
- The approximation $p\theta(x_{t-1}|x_t)$ follows a normal distribution and its mean and variance are set as follows:

$$\begin{cases} \mu_\theta(x_t, t) & := & \tilde{\mu}_t(x_t, x_0) \\ \Sigma_\theta(x_t, t) & := & \tilde{\beta}_t I \end{cases}$$

Loss Function

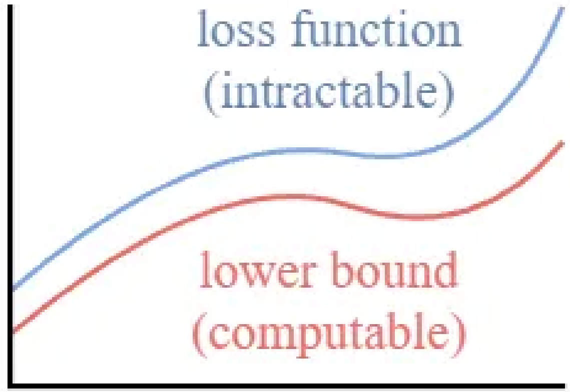
We can define our loss as a Negative Log-Likelihood:

$$Loss = \log p_\theta(x_0)$$

This setup is very similar to the one in VAE. instead of optimizing the intractable loss function itself, we can optimize the Variational Lower Bound.

optimizing a computable lower bound

- This setup is very similar to the one in VAE. instead of optimizing the intractable loss function itself, we can optimize the Variational Lower Bound.



By optimizing a computable lower bound, we can indirectly optimize the intractable loss function.

$$-\log p_\theta(x_0) \le -\log p_\theta(x_0) + D_{\mathrm{KL}}(q(x_{1:T}|x_0)\|p_\theta(x_{1:T}|x_0))$$

$$\vdots$$

$$-\log p_\theta(x_0) \le \boxed{\mathbb{E}_q\left[\log \frac{q(x_{1:T}|x_0)}{p_\theta(x_{0:T})}\right]}$$

Variational Lower Bound

$$\vdots$$

$$-\log p_\theta(x_0) \le \mathbb{E}_q\Big[\underbrace{D_{\mathrm{KL}}(q(x_T|x_0) \,\|\, p_\theta(x_T))}_{L_T} + \sum_{t=2}^{T} \underbrace{D_{\mathrm{KL}}(q(x_{t-1}|x_t, x_0) \,\|\, p_\theta(x_{t-1}|x_t))}_{L_{t-1}} \underbrace{- \log p_\theta(x_0|x_1)}_{L_0}\Big]$$

① ② ③

- No learnable parameters
- Just a Gaussian noise

Constant

⇩

Ignorable

Stepwise denoising term

Reconstruction term

By expanding the variational lower bound, we found that it can be represented with the following three terms:

Training objective for the diffusion model

- Since the reverse diffusion process is not directly computable, we train a neural network $\varepsilon\theta$ to approximate it.
- The training objective (loss function) is as follows:

$$x_t = \sqrt{\bar{a}_t}x_0 + \sqrt{1 - \bar{a}_t}\varepsilon$$

$$L_{simple} = \mathbb{E}_{t,x_0,\varepsilon}[||\varepsilon - \varepsilon_\theta(x_t, t)||^2]$$

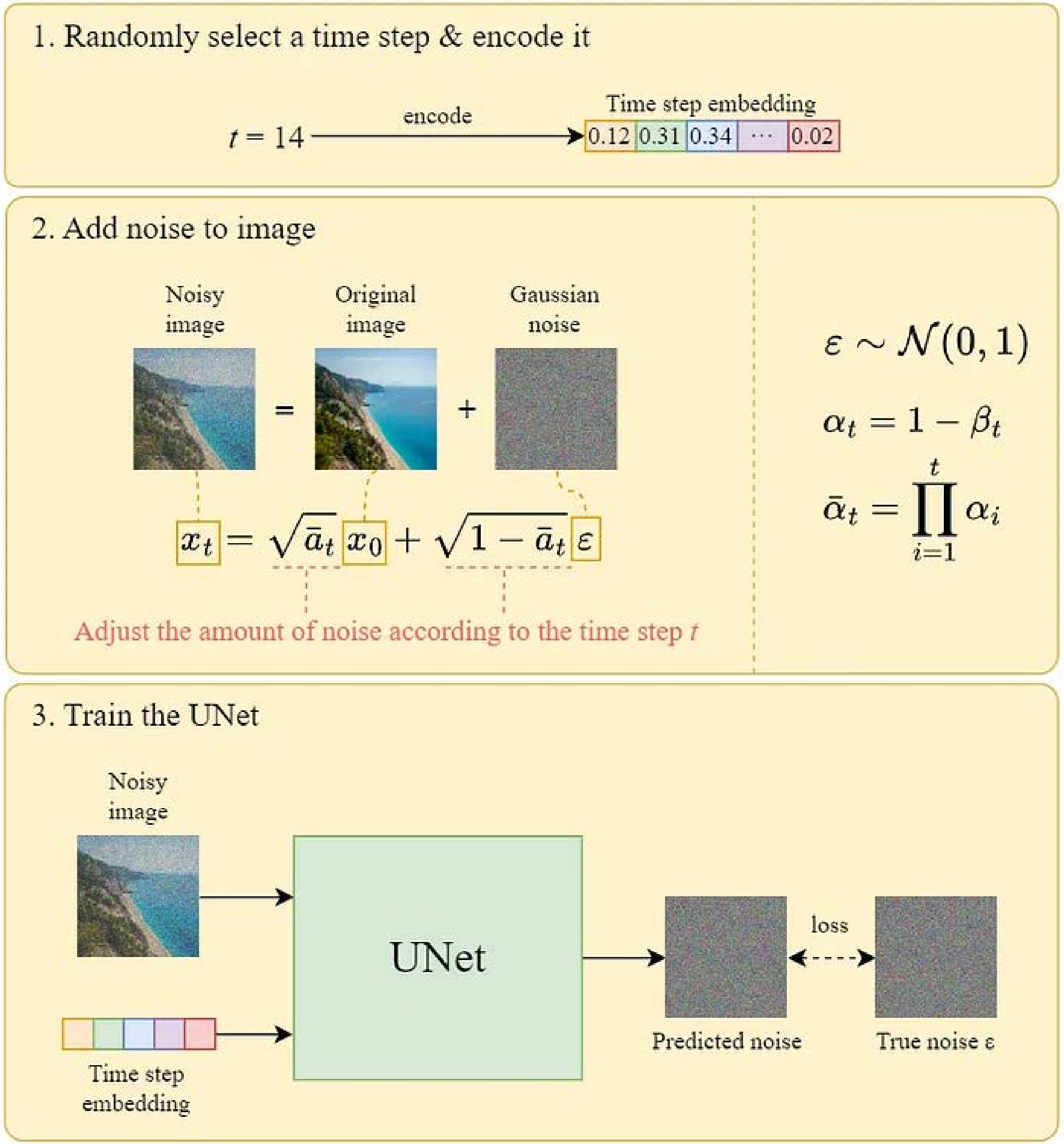
Algorithm 1 Training

1. **Repeat**
2. $x_0 \sim q(x_0)$
3. $t \sim \mathrm{Uniform}(\{1, \ldots, T\})$
4. $\varepsilon \sim \mathcal{N}(0, I)$
5. Take gradient descent step on

$$\nabla_\theta \left\| \epsilon - \epsilon_\theta\left(\sqrt{\bar{a}_t} x_0 + \sqrt{1 - \bar{a}_t} \varepsilon , t\right) \right\|^2$$

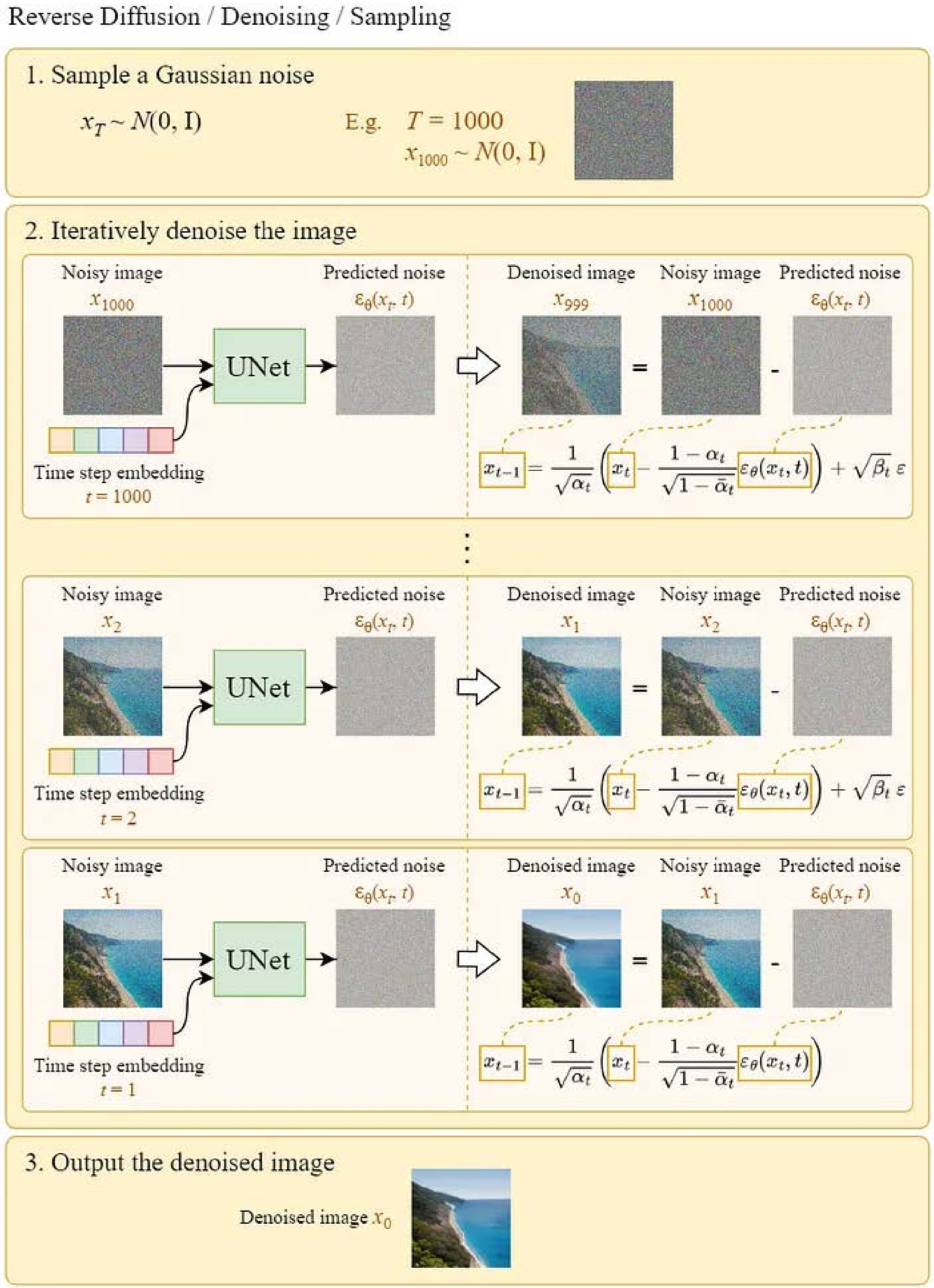
6. **Until** converged

For each training step:



Algorithm 2 Sampling (Reverse Diffusion)

1. $\mathbf{x_T} \sim \mathcal{N}(0, \mathbf{I})$
2. for $t = T, \dots, 1$ do
3. $\quad$ $\mathrm{z} \sim \mathcal{N}(0, \mathbf{I})$ if $t > 1$, else $z = 0$
4. $\quad$ $x_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{a}_t}}\epsilon_\theta(x_t, t)\right) + \sigma_t z$
5. end for
6. return x_0



in the last step, we simply output the learned mean ***μϑ(x₁, 1)*** without adding the noise to it.

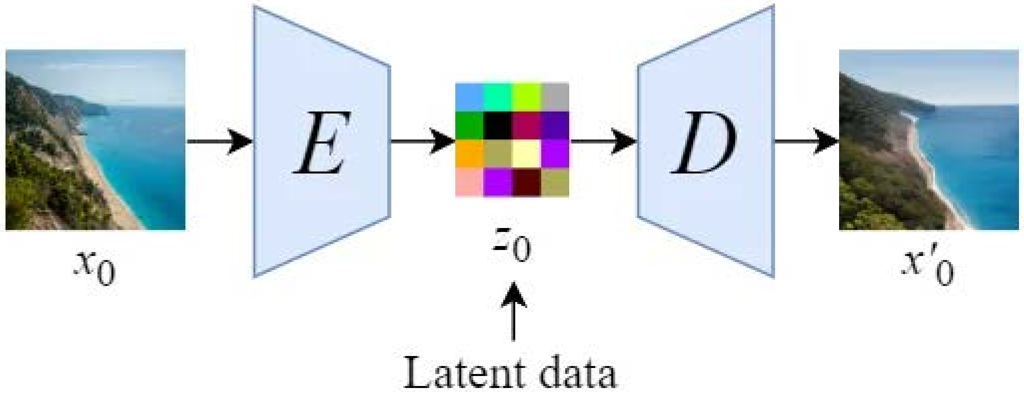
Summary

- The Diffusion model is divided into two parts: forward diffusion and reverse diffusion.
- The forward diffusion can be done using the closed-form formula.
- The backward diffusion can be done using a trained neural network.
- To approximate the desired denoising step q, we just need to approximate the noise ε_t using a neural network $\varepsilon\theta$.
- Training on the simplified loss function yields better sample quality.

Stable Diffusion

How does Stable Diffusion work?

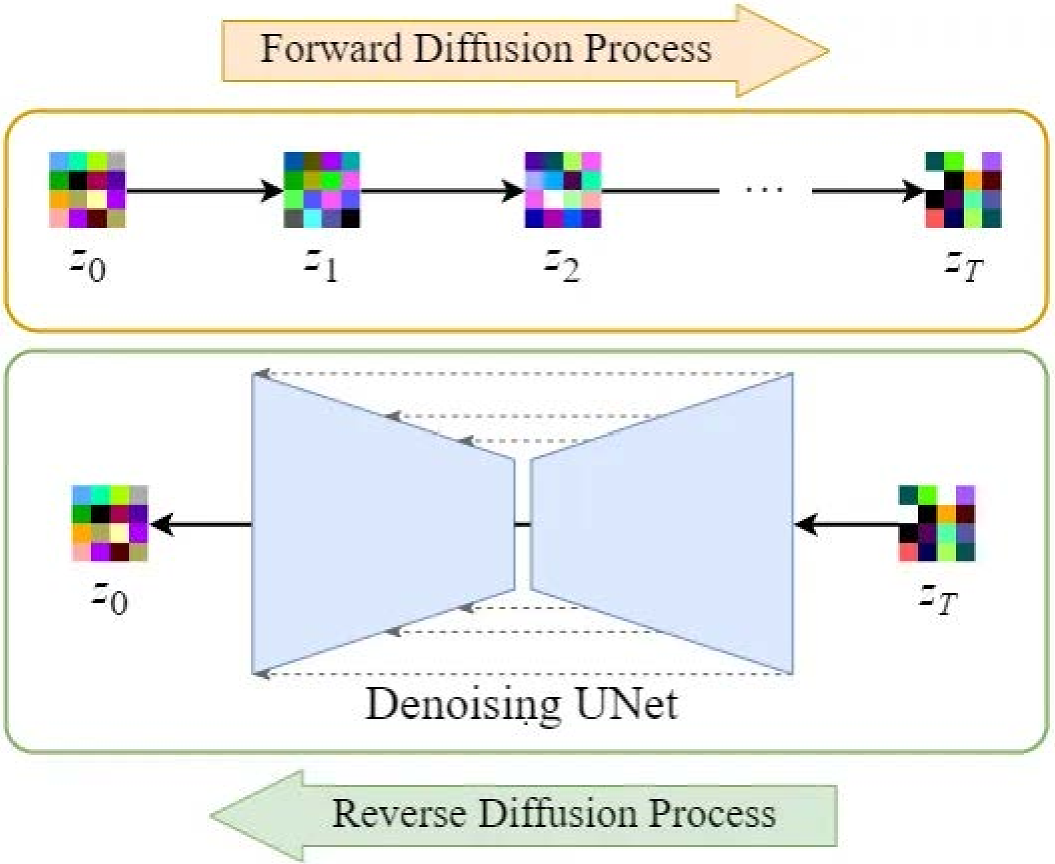
- Stable Diffusion is based on a particular type of diffusion model called Latent Diffusion, proposed in High-Resolution Image Synthesis with Latent Diffusion Models.
- Diffusion models are machine learning systems that are trained to *denoise* random Gaussian noise step by step, to get to a sample of interest, such as an *image.*
- The reverse denoising process is slow because of its repeated, sequential nature. In addition, these models consume a lot of memory because they operate in pixel space.



Departure to Latent Space

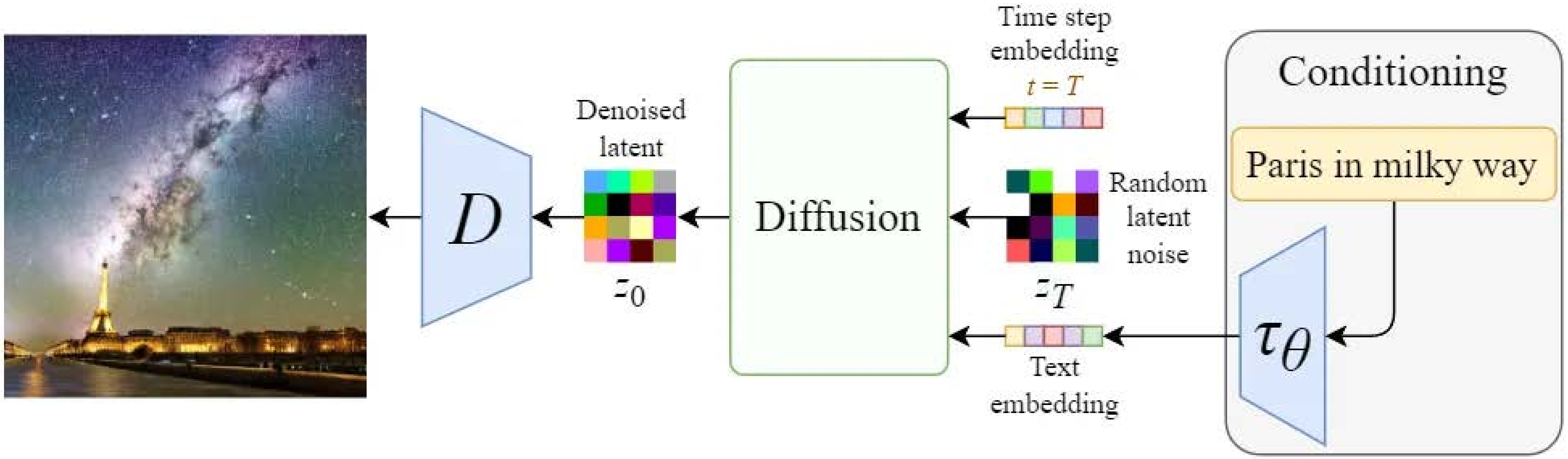
- Train an autoencoder to learn to compress the image data into lower-dimensional representations.
  - By using the trained encoder E, we can encode the full-sized image into lower dimensional latent data (compressed data).
  - By using the trained decoder D, we can decode the latent data back into an image.
- After encoding the images into latent data, the forward and reverse diffusion processes will be done in the latent space.

Latent Diffusion

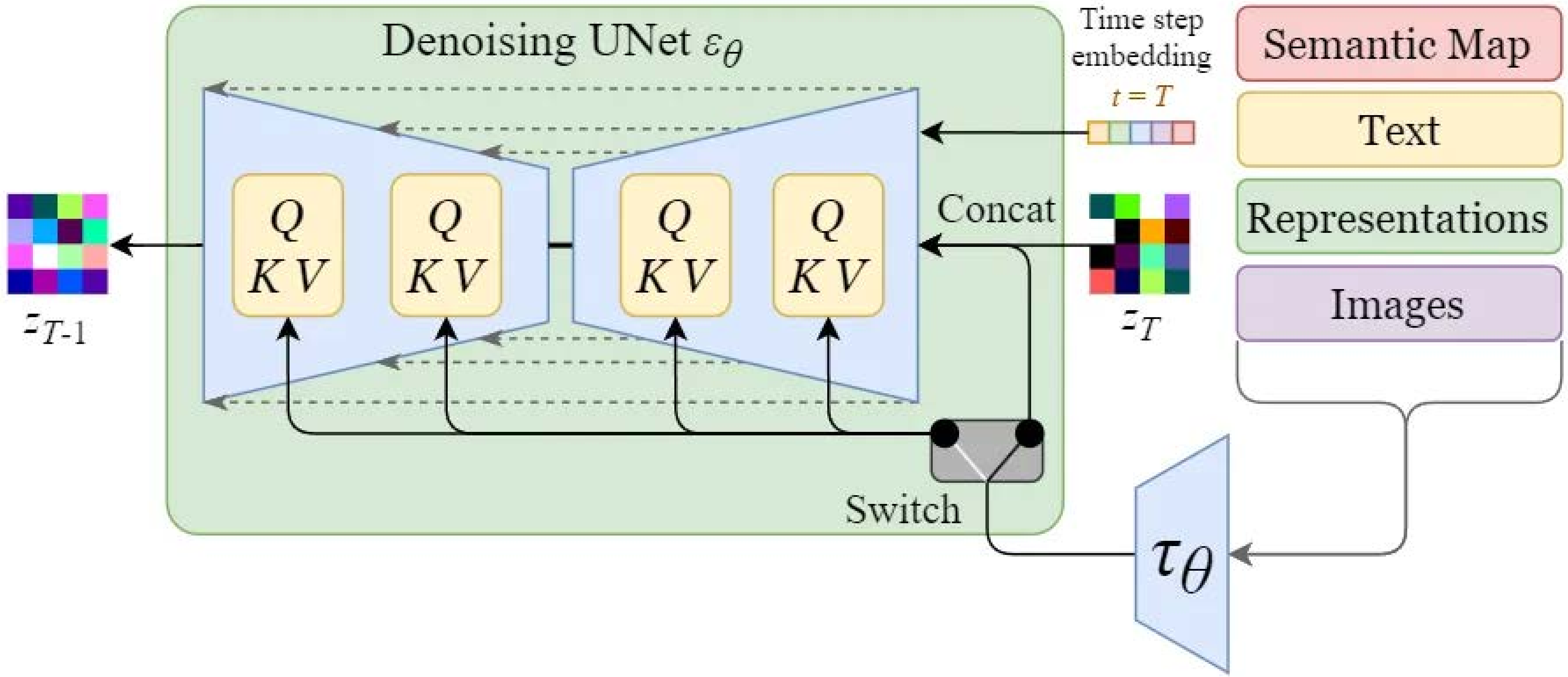


Conditioning

modify the inner diffusion model to accept conditioning inputs.



Conditioning mechanism



The inner diffusion model is turned into a conditional image generator by augmenting its denoising U-Net with the cross-attention mechanism.
The switch is used to control between different types of conditioning inputs

Conditioning

- The switch in the above diagram is used to control between different types of conditioning inputs:
  - For text inputs, they are first converted into embeddings (vectors) using a language model $\tau\theta$ (e.g. BERT, CLIP), and then they are mapped into the U-Net via the (multi-head) Attention(Q, K, V) layer.
  - For other spatially aligned inputs (e.g. semantic maps, images, inpainting), the conditioning can be done using concatenation.

Training in Stable diffusion

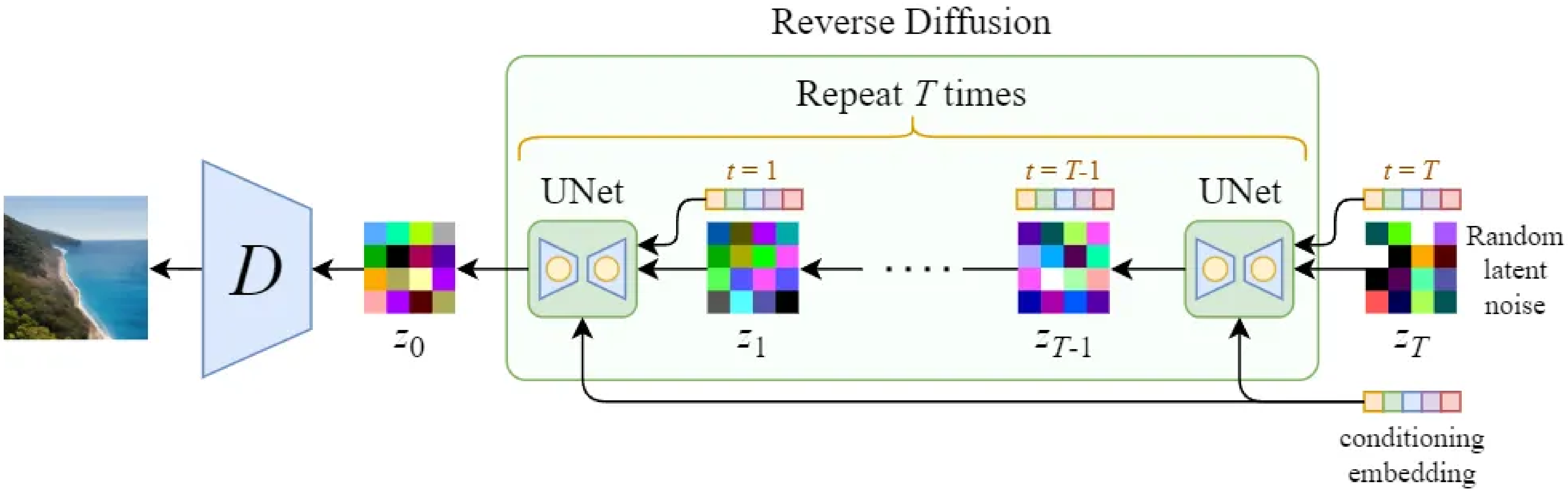
$$\boxed{z_0 = E(x_0)}$$

$$\boxed{z_t = \sqrt{\bar{\alpha}_t}\,\boxed{z_0} + \sqrt{1-\bar{\alpha}_t}\,\varepsilon}$$

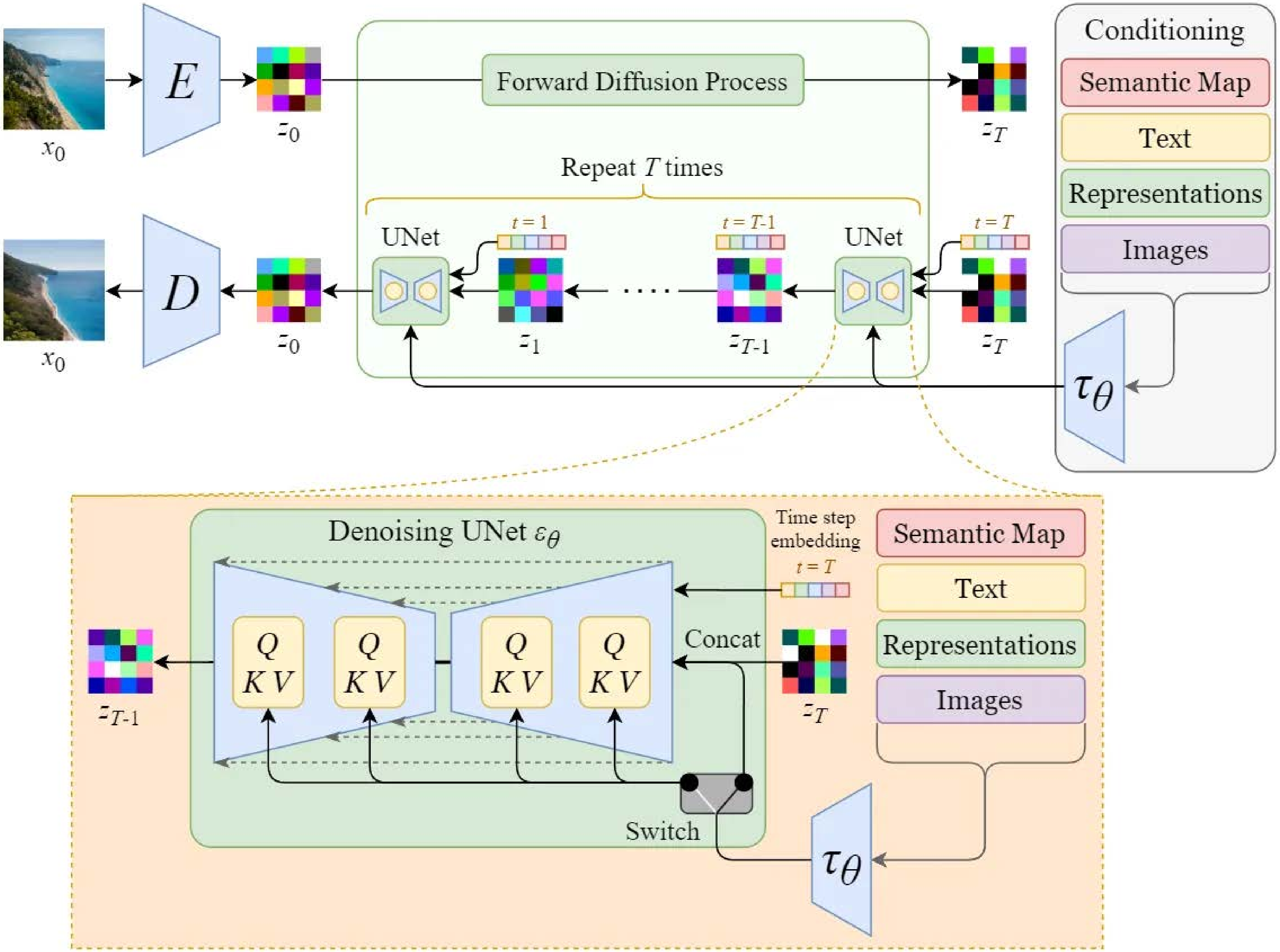
$$L_{\mathrm{LDM}} = \mathbb{E}_{t,z_0,\varepsilon,y}\Big[\|\boldsymbol{\varepsilon} - \boldsymbol{\varepsilon}_\theta(\boxed{z_t}, t, \underbrace{\boxed{\tau_\theta(y)}}_{\text{Conditioning}})\|^2\Big]$$

- Input latent data z_t instead of the image x_t.
- Added conditioning input $\tau\theta(y)$ to the U-Net.

Sampling in Stable diffusion



Since the size of the latent data is much smaller than the original images, the denoising process will be much faster.



x_0
E
z_0
Forward Diffusion Process
z_T
Conditioning
Semantic Map
Text
Representations
Images
Repeat T times
UNet
$t = 1$
$t = T\text{-}1$
UNet
$t = T$
D
z_0
z_1
$z_{T\text{-}1}$
z_T
x_0
τ_θ
Denoising UNet ε_θ
Time step
embedding
$t = T$
Semantic Map
Text
Representations
Images
Q
K V
Q
K V
Q
K V
Q
K V
Concat
$z_{T\text{-}1}$
z_T
Switch
τ_θ

Latent diffusion model

(LDM; Rombach & Blattmann, et al. 2022)

- Runs the diffusion process in the latent space instead of pixel space, making training cost lower and inference speed faster.
- The paper explored two types of regularization in autoencoder training to avoid arbitrarily high-variance in the latent spaces.
    1. KL-reg: A small KL penalty towards a standard normal distribution over the learned latent, similar to VAE.
    2. VQ-reg: Uses a vector quantization layer within the decoder
- The diffusion and denoising processes happen on the latent vector
- The denoising model is a time-conditioned U-Net, augmented with the cross-attention mechanism to handle flexible conditioning information for image generation (e.g. class labels, semantic maps, blurred variants of an image).

Latent Diffusion

- in latent diffusion the model is trained to generate latent (compressed) representations of the images. Latent diffusion can reduce the memory and compute complexity by applying the diffusion process over a lower dimensional latent space.
- There are three main components in latent diffusion.
  1. An autoencoder (VAE).
  2. A U-Net.
  3. A text-encoder, *e.g.* CLIP's Text Encoder.

1. The autoencoder (VAE)

- The VAE model has two parts, an encoder and a decoder.
  - The encoder is used to convert the image into a low dimensional latent representation, which will serve as the input to the *U-Net* model.
  - The decoder transforms the latent representation back into an image.
  - During latent diffusion *training*, the encoder is used to get the latent representations (*latents*) of the images for the forward diffusion process, which applies more and more noise at each step.
  - During *inference*, the denoised latents generated by the reverse diffusion process are converted back into images using the VAE decoder.

As we will see during inference we **only need the VAE decoder**.

The U-Net

- The U-Net has an encoder part and a decoder part both comprised of ResNet blocks.
  - The encoder compresses an image representation into a lower resolution image representation.
  - The decoder decodes the lower resolution image representation back to the original higher resolution image representation.
- To prevent the U-Net from losing important information while downsampling, short-cut connections are usually added between the downsampling ResNets of the encoder to the upsampling ResNets of the decoder.
- Additionally, the stable diffusion U-Net is able to condition its output on text-embeddings via cross-attention layers. The cross-attention layers are added to both the encoder and decoder part of the U-Net usually between ResNet blocks.

3. The Text-encoder

- The text-encoder is responsible for transforming the input prompt, e.g. "An astronaut riding a horse" into an embedding space that can be understood by the U-Net.
- It is usually a simple transformer-based encoder that maps a sequence of input tokens to a sequence of latent text-embeddings.

Stable Diffusion during inference