POLITECNICO
MILANO 1863

Advanced Deep Learning

- Image Generation with DDPM -

Matteo Matteucci, PhD (matteo.matteucci@polimi.it)
Artificial Intelligence and Robotics Laboratory
Politecnico di Milano

AIRLAB
ARTIFICIAL INTELLIGENCE AND ROBOTICS LAB

Lecture Objectives

Starting from the concept of Image Generation and its evolution, this lecture focuses on Denoising Diffusion Probabilistic Models (DDPM). These are a class of latent variable models inspired by nonequilibrium thermodynamics. Nowadays DDPM have become the de facto standard in synthetic image (and video) generation.

Denoising Diffusion Probabilistic Models

Jonathan Ho
UC Berkeley
jonathanho@berkeley.edu

Ajay Jain
UC Berkeley
ajayj@berkeley.edu

Pieter Abbeel
UC Berkeley
pabbeel@cs.berkeley.edu

Abstract

We present high quality image synthesis results using diffusion probabilistic models, a class of latent variable models inspired by considerations from nonequilibrium thermodynamics. Our best results are obtained by training on a weighted variational bound designed according to a novel connection between diffusion probabilistic models and denoising score matching with Langevin dynamics, and our models naturally admit a progressive lossy decompression scheme that can be interpreted as a generalization of autoregressive decoding. On the unconditional CIFAR10 dataset, we obtain an Inception score of 9.46 and a state-of-the-art FID score of 3.17. On 256x256 LSUN, we obtain sample quality similar to ProgressiveGAN. Our implementation is available at https://github.com/hojonathanho/diffusion.

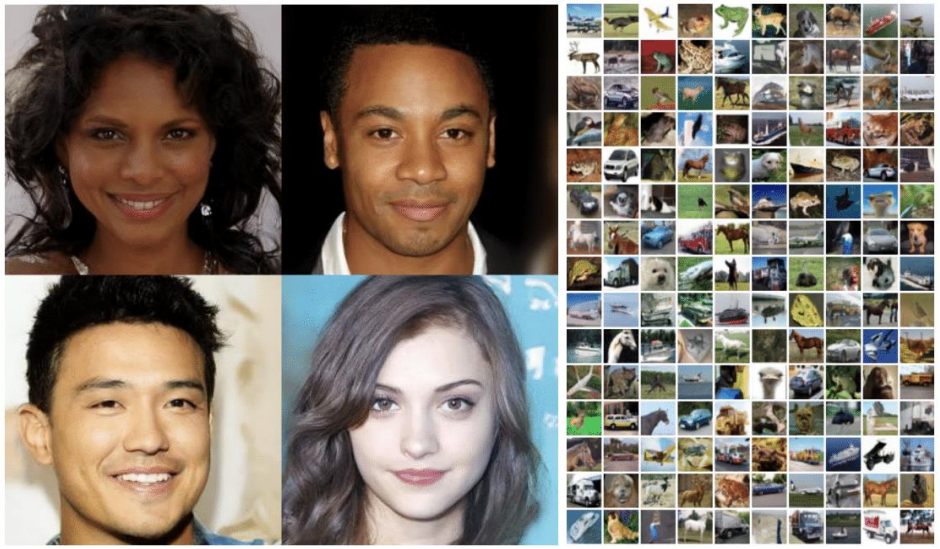
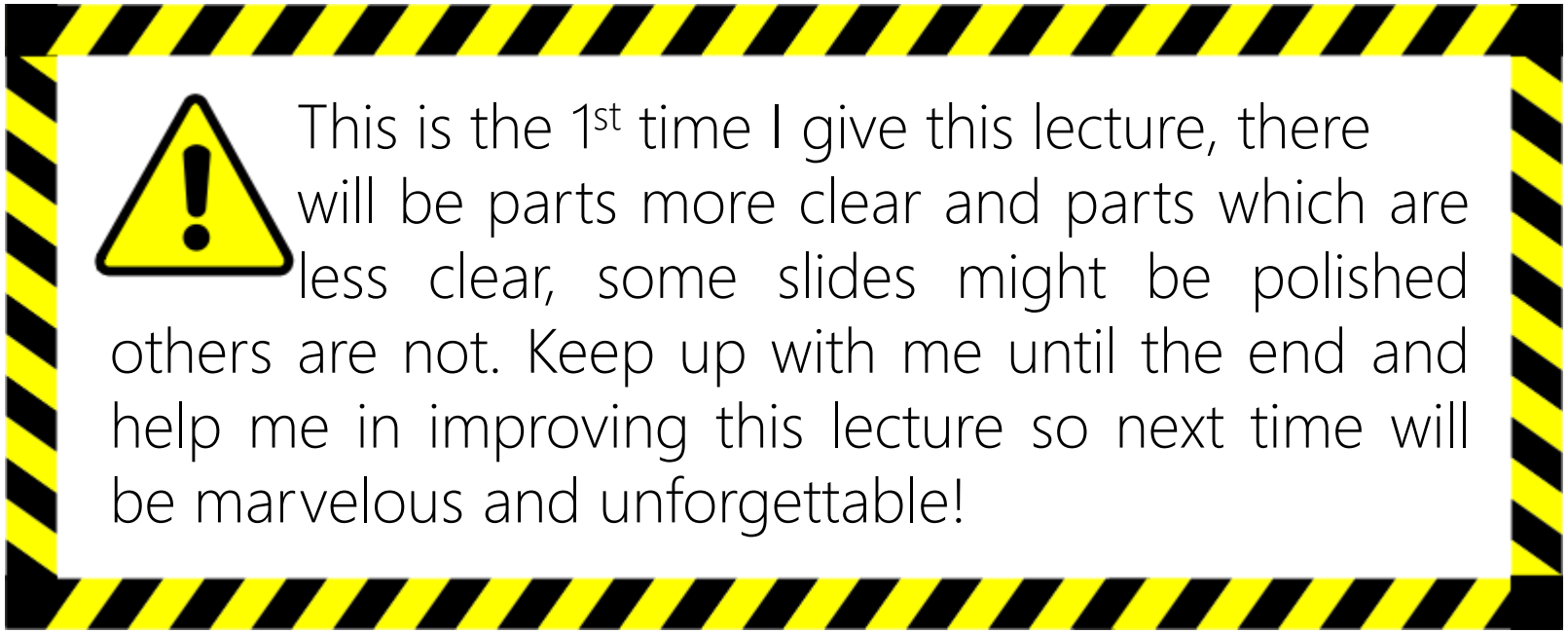


Figure 1: Generated samples on CelebA-HQ 256 × 256 (left) and unconditional CIFAR10 (right)

Lecture Objectives

Starting from the concept of Image Generation and its evolution, this lecture focuses on Denoising Diffusion Probabilistic Models (DDPM). These are a class of latent variable models inspired by nonequilibrium thermodynamics. Nowadays DDPM have become the de facto standard in synthetic image (and video) generation.



Today's Special

Do you know who is this guy?

Introduction to Image Generation

- Brief History of Image Generation
- The Image Generation Trilemma
- Models for Image generation

Denoising Diffusion Probabilistic Models

- Idea and model behing DDPM
- Class guided and classifier free generation
- The basic DDPM architecture (U-Net)
- Techniques for speeding up DDPM

A python notebook making sense of this all …

https://thispersondoesnotexist.com/

Karras, Tero, et al. "Analyzing and improving the image quality of stylegan." *CVPR 2020*

https://thesecatsdonotexist.com

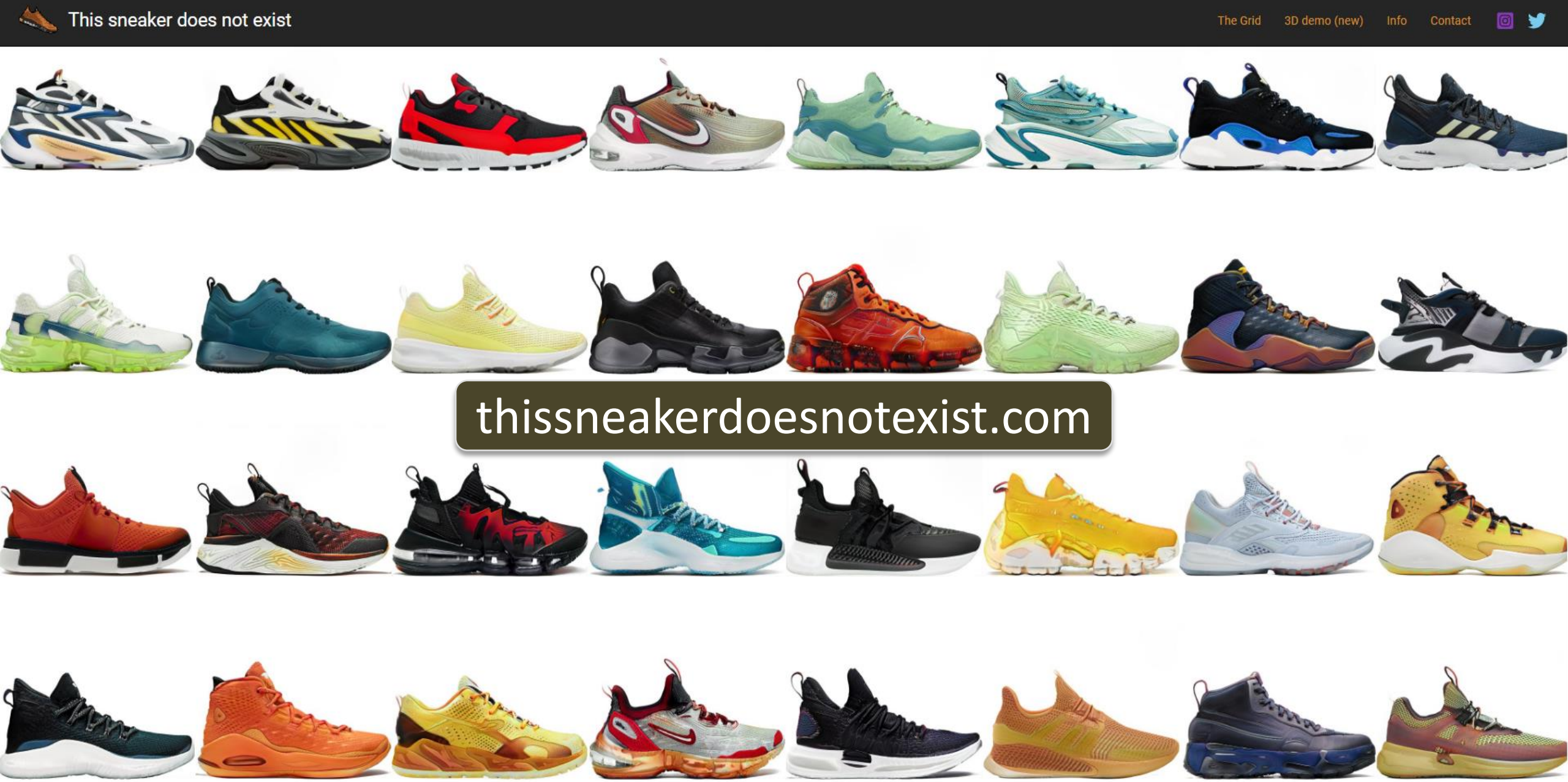
Karras, Tero, et al. "Analyzing and improving the image quality of stylegan." *CVPR 2020*

https://thisnightskydoesnotexists.com

Karras, Tero, et al. "Analyzing and improving the image quality of stylegan." *CVPR 2020*

https://thisponydoesnotexists.com

Karras, Tero, et al. "Analyzing and improving the image quality of stylegan." *CVPR 2020*



Karras, Tero, et al. "Analyzing and improving the image quality of stylegan." CVPR 2020

Sneaker Editor

Use the sliders below the image to edit the sneaker

Normal Futuristic

Low creativity High creativity

Lighter color Darker color

Return to grid

Sneaker Editor

Use the sliders below the image to edit the sneaker

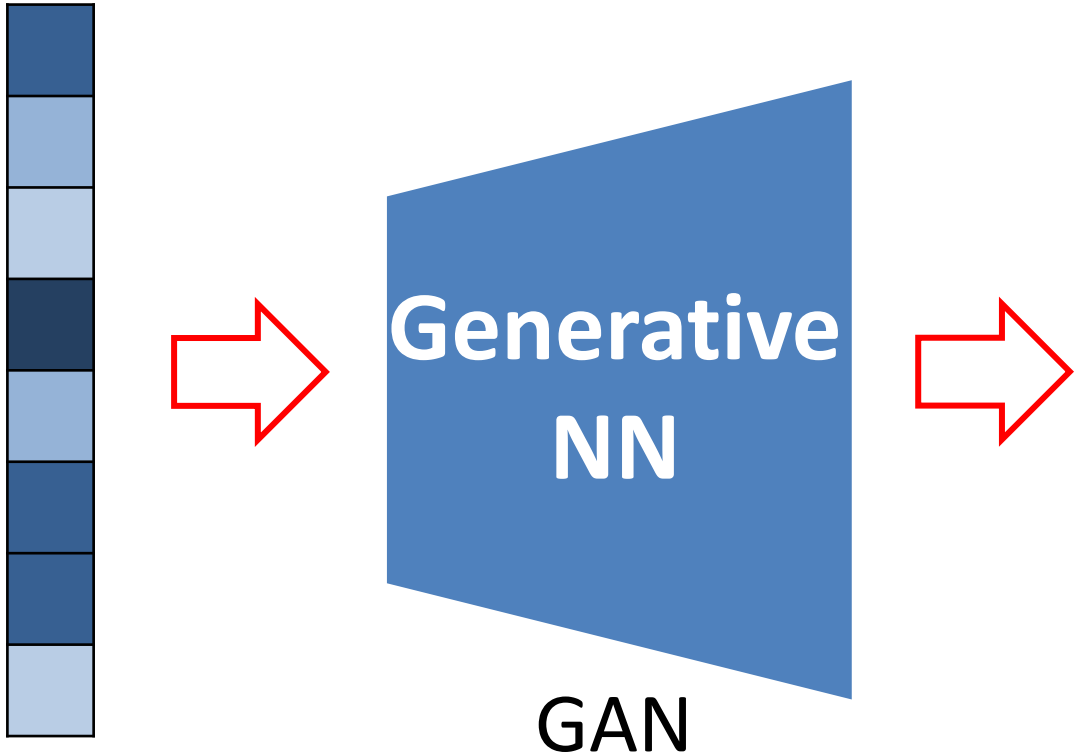
Normal Futuristic

Low creativity High creativity

Lighter color Darker color

Return to grid

What is Image Generation?

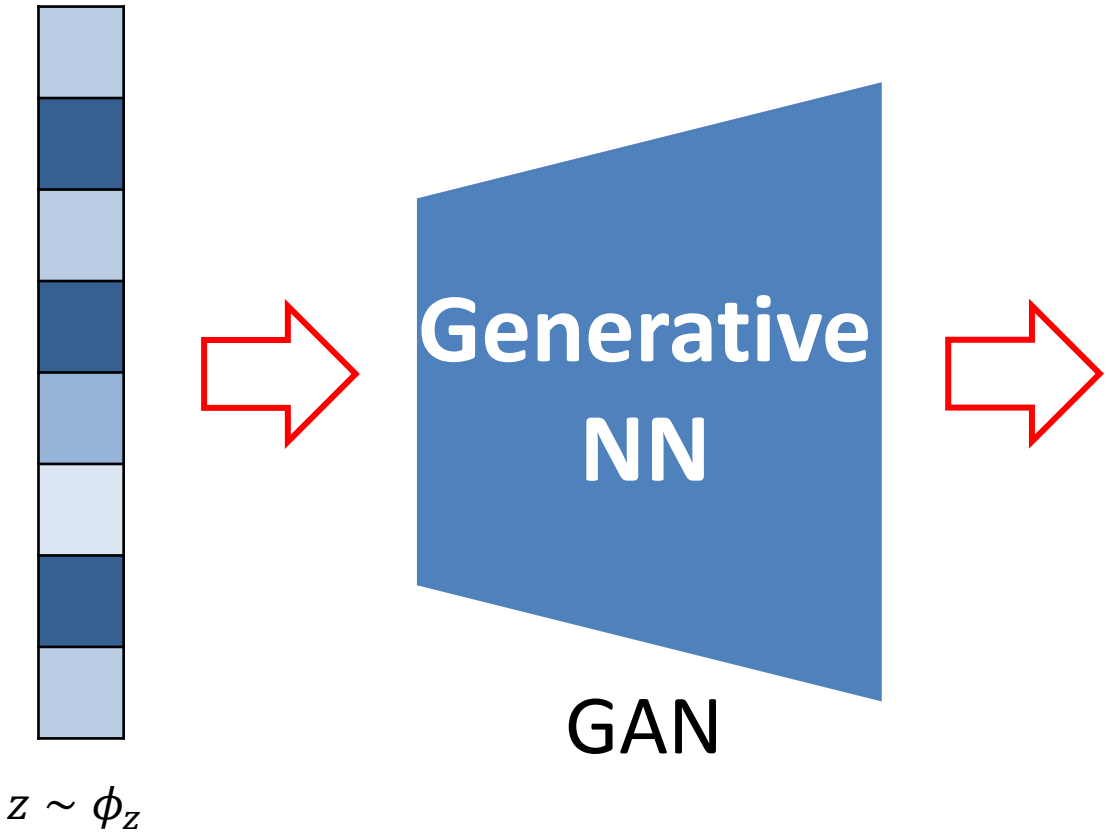


$z \sim \phi_z$

Random vector z
from some distribution ϕ_z

Karras, Tero, et al. "Analyzing and improving the image quality of stylegan." *CVPR 2020*

What is Image Generation?



Random vector z
from some distribution ϕ_z

Karras, Tero, et al. "Analyzing and improving the image quality of stylegan." *CVPR 2020*

When it deals with text we call it Generative AI …

```
>> I want the
proof in verse
form that there
are infinite
prime numbers
```

ChatGPT3

Oh, I see! A proof in verse form you say,
I'll try to make it elegant in every way.
Let p1, p2, p3, …, pn be our list of primes,
Of which there are infinitely many, it's no crime.
Suppose, for the sake of contradiction, not true,
That our list of primes has only a finite few.
Let P be the product of all primes we have seen,
And add one to P, call it our number N, quite keen.
Since N is not divisible by any of the primes,
It must be divisible by a prime that surpasses our finds.
This contradicts our assumption, that the list of primes is finite,
So we must conclude, the list goes on, and will never diminish.
Thus we have shown, in this poetic proof,
That the primes go on, it's a mathematical proof

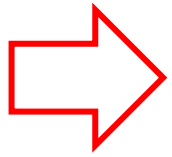
Slide curtesy of Prof. Giacomo Boracchi

… we can combine the two …

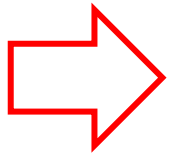
```
>> Pope Francis
wearing a long
white puffer coat
```

+

$z \sim \phi_z$



Midjourney



Random vector z
from some distribution ϕ_z

… and generate many!

```
>> Pope Francis
wearing a long
white puffer coat
```

+

$z \sim \phi_z$

Midjourney

Random vector z
from some distribution ϕ_z

Why do we need to generate images?

Beside learning "holy grail" of natural images distribution (manifold), data generators provide:

- New data augmentations techniques
- Privacy preserving datasets (e.g., people faces)
- Solutions to inverse problems (e.g., super-resolution, inpainting, colorization, etc.)
- Latent representations useful as general features
- Regularization priors in computer vision problems or to perform anomaly detection
- …
- Visual content generation and realistic samples for artwork!

Radford, et al "Unsupervised representation learning with deep convolutional generative adversarial networks. ICLR 2016

Content it has never seen before!

Image Generation in 2014, image size: 32x32

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2014). Generative adversarial nets. Advances in neural information processing systems, 27.

Image Generation in 2014, image size: 32x32

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2014). Generative adversarial nets. Advances in neural information processing systems, 27.

Image Generation in 2016, image size: 224x224

Radford, A., Metz, L., & Chintala, S. Unsupervised representation learning with deep convolutional generative adversarial networks. ICLR 2016

Image Generation in 2018

Karras, Tero, et al. "Progressive growing of gans for improved quality, stability, and variation." ICLR 2018

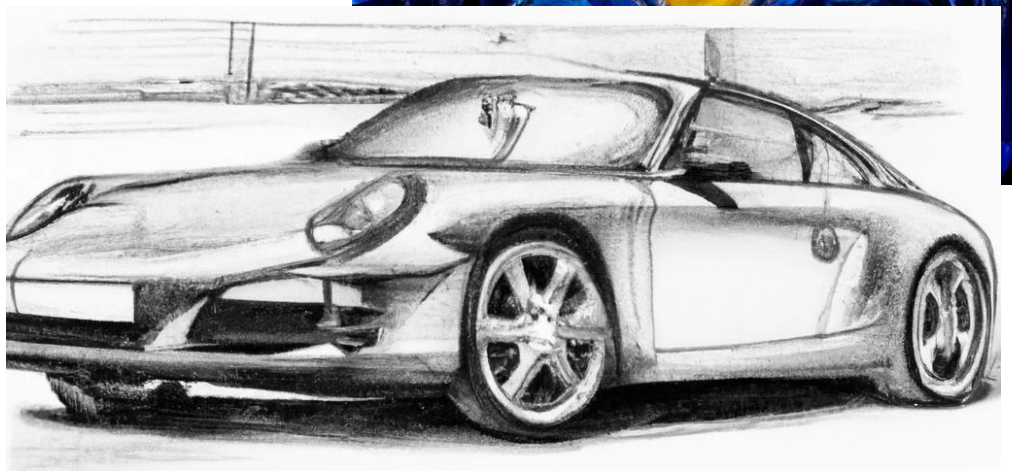
Image Generation in 2019, image size: 1000x1000

Tero Karras, Samuli Laine, Timo Aila «A Style-Based Generator Architecture for Generative Adversarial Networks" CVPR 2019

Dall-E 2, 2021

A van Gogh style painting of an American football player

A photo of a white fur monster standing in a purple room



A hand drawn sketch of a Porsche 911

A handpalm with a tree growing on top of it

A. Radford, et al. "Learning transferable visual models from natural language supervision." ICML 2021.

Sora, 2024 Video Generator

https://openai.com/index/sora/

nighttime footage of a hermit crab using an incandescent lightbulb as its shell

Photorealistic closeup video of two pirate ships battling each other as they sail inside a cup of coffee.

Several giant wooly mammoths approach treading through a snowy meadow, their long wooly fur lightly blows in the wind as they walk, [..]

POLITECNICO MILANO 1863

Lil'Log | What are Diffusion Models?

Date: July 11, 2021 | Estimated Reading Time: 32 min | Author: Lilian Weng

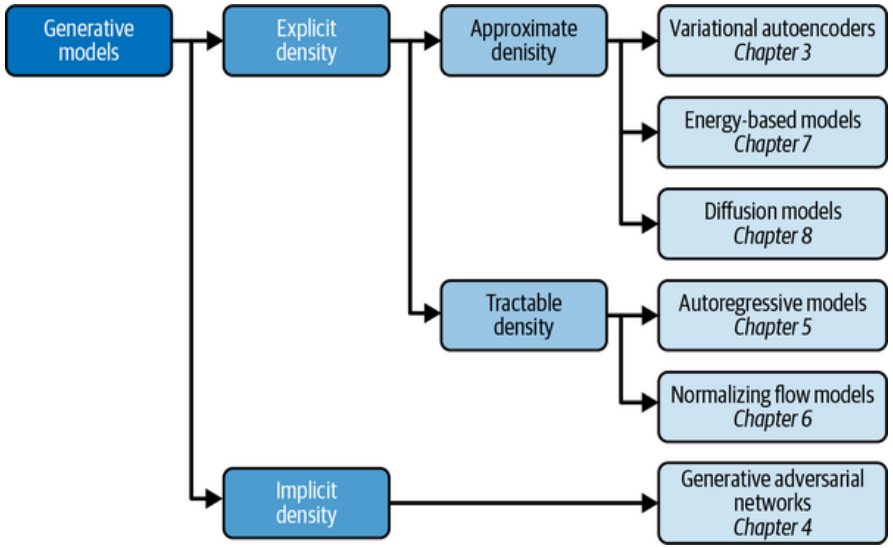
https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

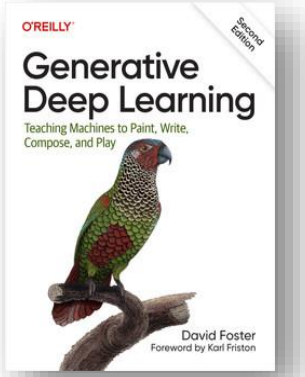
What are Diffusion Models?

Matteo Matteucci, PhD (matteo.matteucci@polimi.it)
Artificial Intelligence and Robotics Laboratory
Politecnico di Milano

AIRLAB
ARTIFICIAL INTELLIGENCE AND ROBOTICS LAB

Generative Models Taxonomies

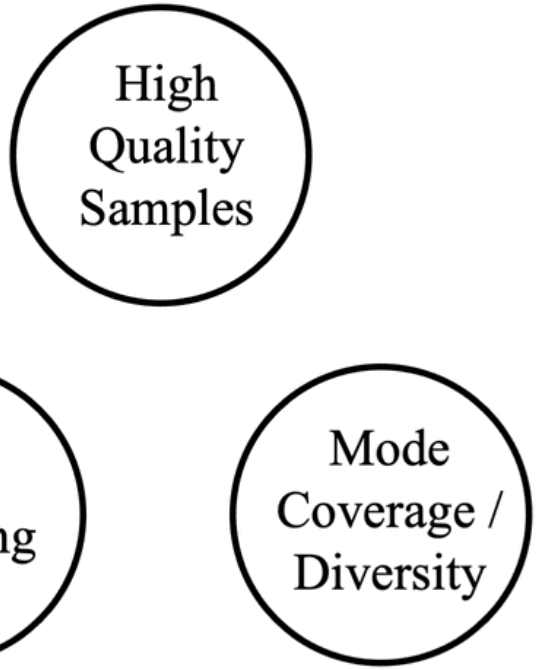




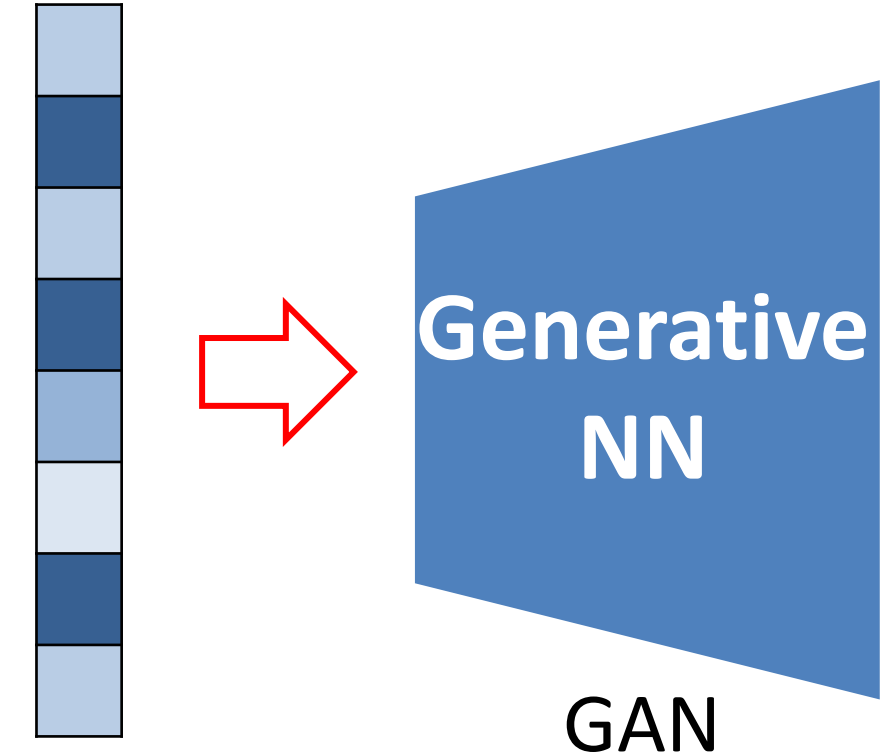
Explicit Density	**Approximate**	Variational Inference	Variational Autoencoder
		Markov chain	Deep Belief Network Restricted Boltzmann Machine
	Tractable	Full Visible Belief Net	NADE MADE PixelRNN/CNN
		Change of variable models	Nonlinear ICA
Implicit Density	**Direct**	Generative Adversarial Network	Minimax GAN Non-saturating GAN GAN variants
		Generative Moment Matching Network	GMMN
	Markov	Generative Stochastic Network	GSN

Thanks NVIDIA for teaching me new words!

The Generative Learning Trilemma



Recall Image Generation …



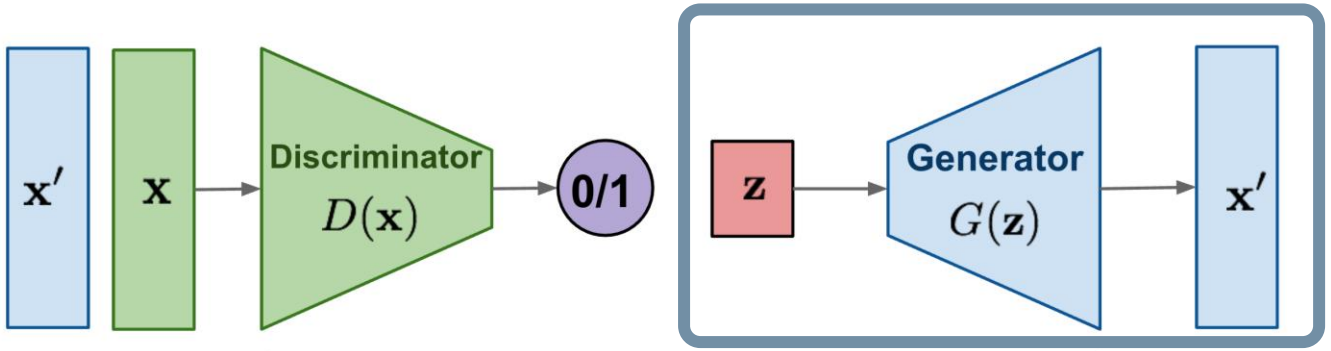
$z \sim \phi_z$

Random vector z
from some distribution ϕ_z

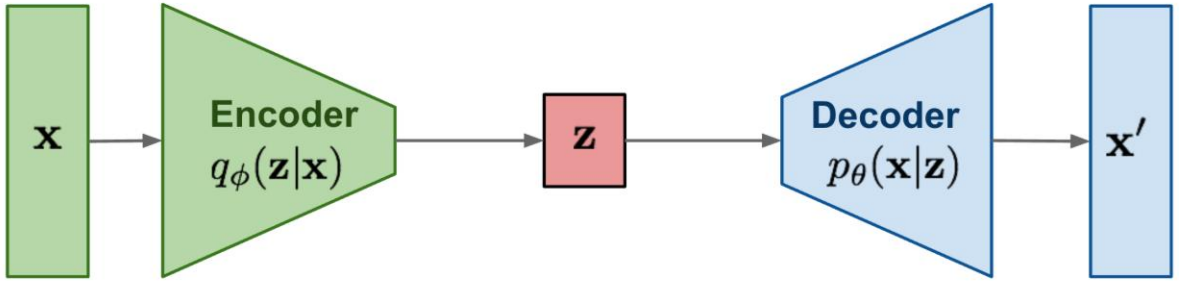
Karras, Tero, et al. "Analyzing and improving the image quality of stylegan." *CVPR 2020*

Generative Models

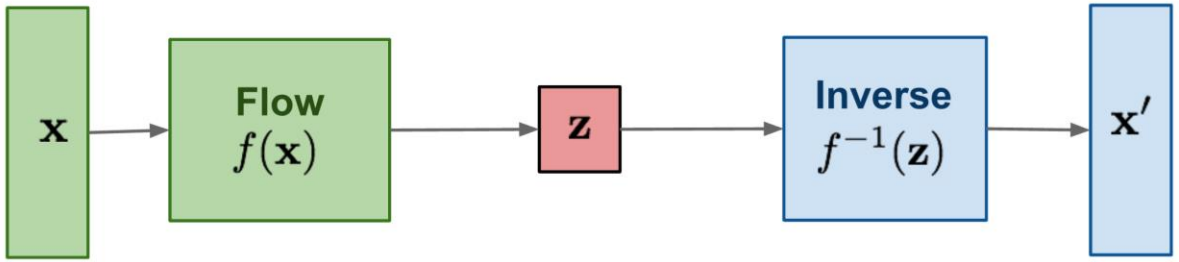
GAN: Adversarial training



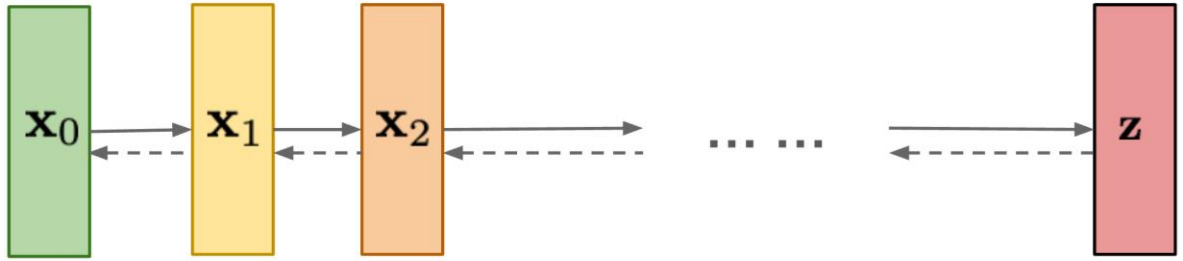
VAE: maximize variational lower bound



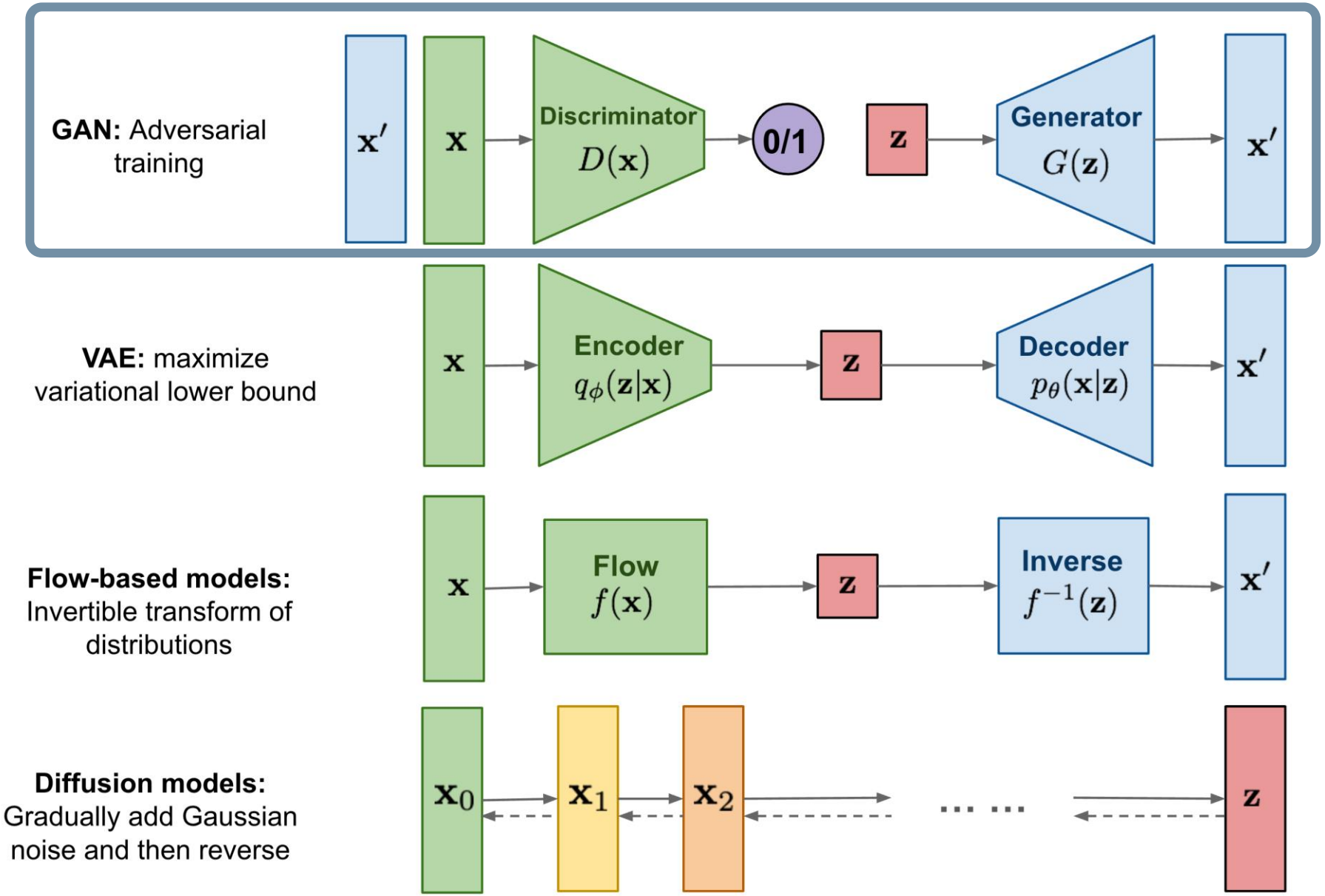
Flow-based models: Invertible transform of distributions



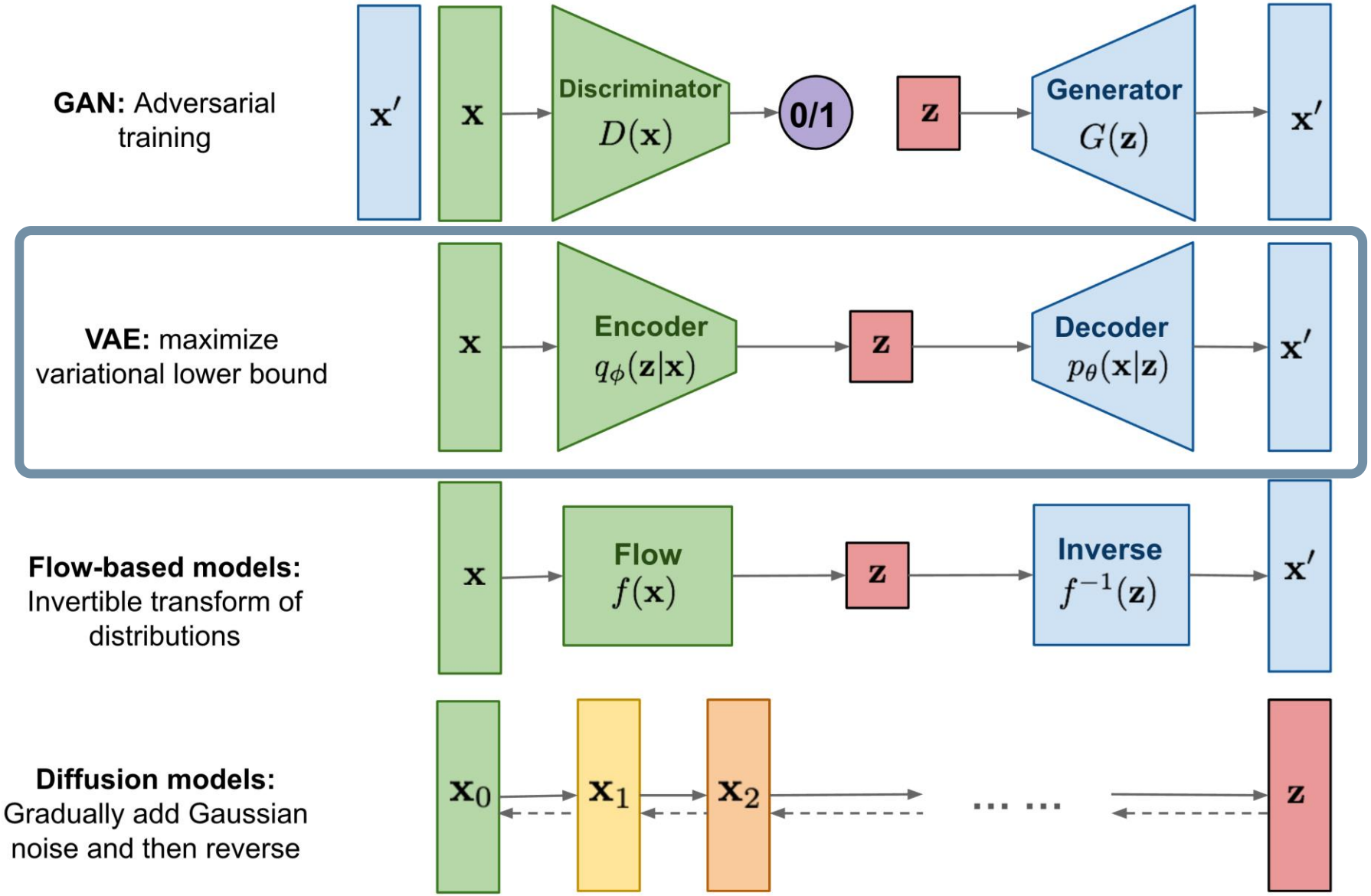
Diffusion models: Gradually add Gaussian noise and then reverse



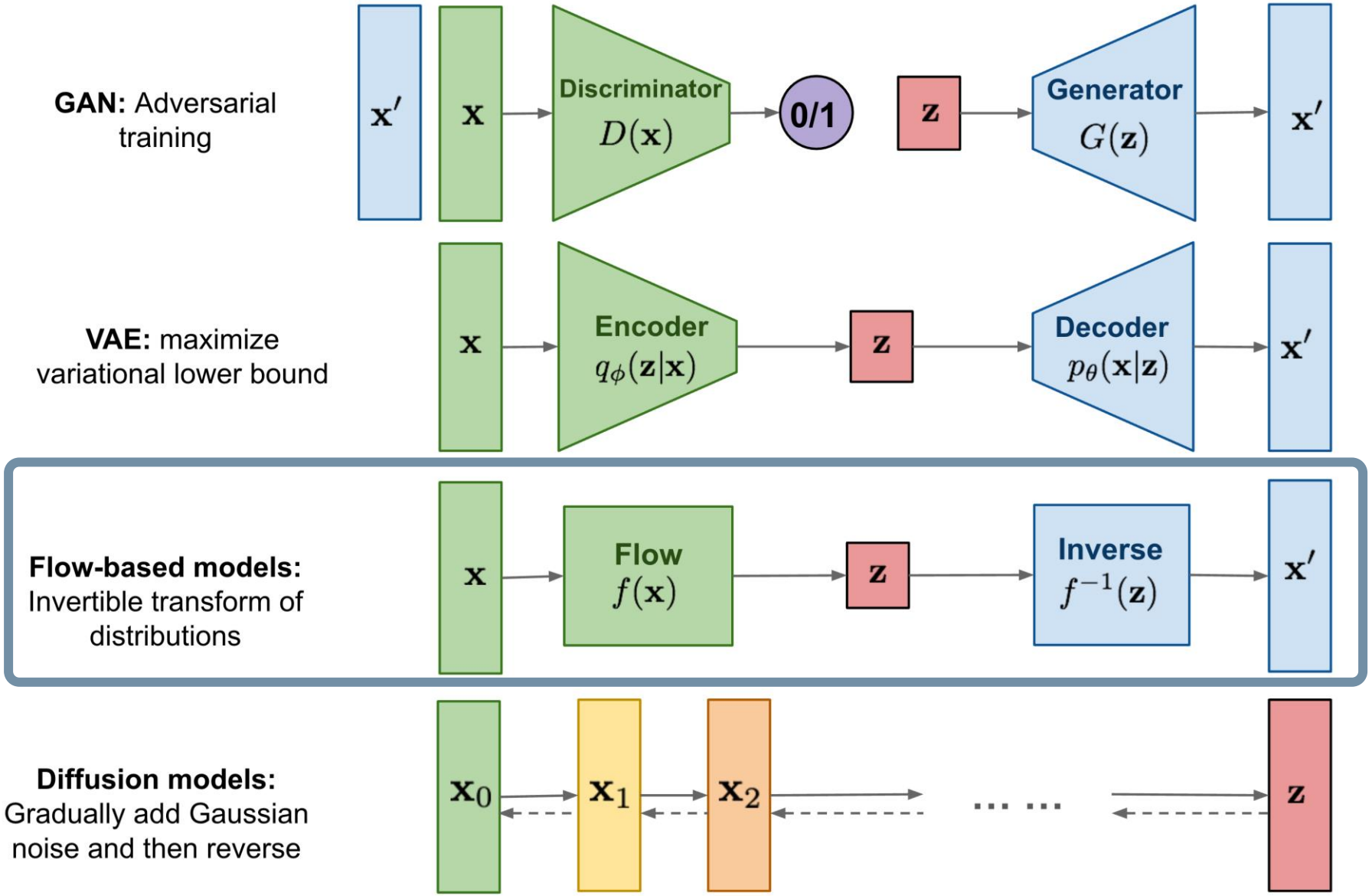
Generative Models



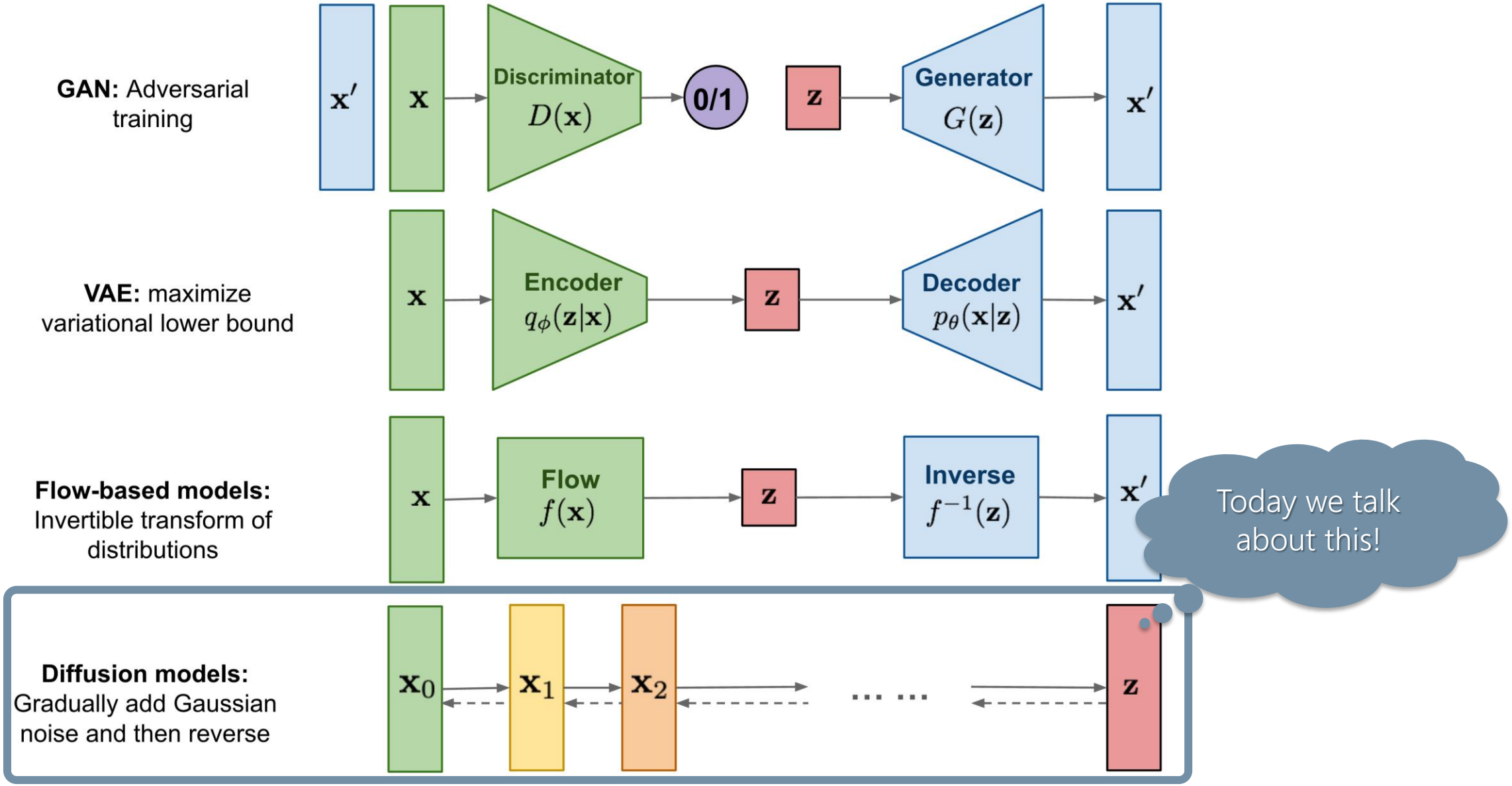
Generative Models



Generative Models



Generative Models



Forward Diffusion Process

Given a data point from a *real data distribution* $x_0 \sim q(x)$ add Gaussian noise in T steps, producing a sequence of noisy samples $x_1, x_2, \ldots, x_T$

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I})$$

Data $\sim p(x)$

Noise$\approx \mathcal{N}(0,1)$

- Step sizes are controlled by a variance schedule $\{\beta_t \in (0,1)\}_{t=1}^{T}$
- Given sufficiently large T and a well-behaved schedule of β_t, the latent x_T is nearly an isotropic Gaussian distribution.

Forward Diffusion Process "Nice Property"

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I})$$

Data $\sim p(x)$

Noise$\approx \mathcal{N}(0,1)$

We can sample $\boldsymbol{x_t}$ at any arbitrary time step $\boldsymbol{t}$ in a closed form using reparameterization trick. Let $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$:

$$x_t = \sqrt{\alpha_t}x_{t-1} + \sqrt{1-\alpha_t}\epsilon_{t-1} = \sqrt{\alpha_t\alpha_{t-1}}x_{t-2} + \boxed{\sqrt{1-\alpha_t\alpha_{t-1}}\bar{\epsilon}_{t-2}}$$
$$= \cdots = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$$

from this we get: $q(x_t|x_0) = N\left(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\overline{\alpha_t})I\right)$

Note: $\bar{\epsilon}_{t-2}$ *merges two Gaussians with different variance* $N(0, \sigma_1^2 I)$ *and* $N(0, \sigma_2^2 I)$ *into* $N(0, (\sigma_1^2 + \sigma_2^2)I)$
in this case the merged standard deviation is $\sqrt{(1-\alpha_t) + \alpha_t(1-\alpha_{t-1})} = \sqrt{1-\alpha_t\alpha_{t-1}}$

Reverse Diffusion Process

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I})$$

Data $\sim p(x)$

Noise$\approx \mathcal{N}(0,1)$

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \boldsymbol{\sigma}_t^2\mathbf{I})$$

If we reverse the above process via $q(x_{t-1}|x_t)$ we could recreate the true sample from $\mathbf{x_0}\sim\mathbf{N(0,I)}$, but this it is unknown, thus **we learn $\boldsymbol{p_\theta(x_{t-1}|x_t)}$** to approximate it (*note: if β_t is small enough, $q(x_{t-1}|x_t)$ is Gaussian*):

$$p_\theta(x_{0:T}) = p(x_T)\prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)$$

$$p_\theta(x_{t-1}|x_t) = N(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t) = \sigma_t^2 I)$$

Common formulations only learn $\mu_\theta(x_t, t)$, fixing $\sigma_t^2 = \beta_t$

Training the Reverse Diffusion Process

$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I})$

Data $\sim p(x)$

Noise$\approx \mathcal{N}(0,1)$

$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \boldsymbol{\sigma}_t^2\mathbf{I})$

To approximate $p_\theta(x_{t-1}|x_t) = N(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$ we train μ_θ to predict $\mu_t = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_t\right)$, and having available x_t at training time we reparametrize ϵ_t as a function of x_t

We learn the noise added to the image at time *t*

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

Training the Reverse Diffusion Process

We learn the noise added to the image at time *t*

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

The process $p_\theta(x_{t-1}|x_t) = N(x_{t-1}; \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t)\right), \Sigma_\theta(x_t, t))$ is trained to minimizes the difference from the observed sample

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

Training the Reverse Diffusion Process

We learn the noise added to the image at time *t*

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

The process $p_\theta(x_{t-1}|x_t) = N(x_{t-1}; \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t)\right), \Sigma_\theta(x_t, t))$ is trained to minimizes the difference from the observed sample, which empirically, Ho et al. (2020) found to work better in the simplified form:

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

Training vs Inference in Denoising Diffusion Processes

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I})$$

Data $\sim p(x)$

Noise$\approx \mathcal{N}(0,1)$

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \boldsymbol{\sigma}_t^2\mathbf{I})$$

The training and sampling/inference algorithms become (Ho et al. 2020)

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** $\mathbf{x}_0$

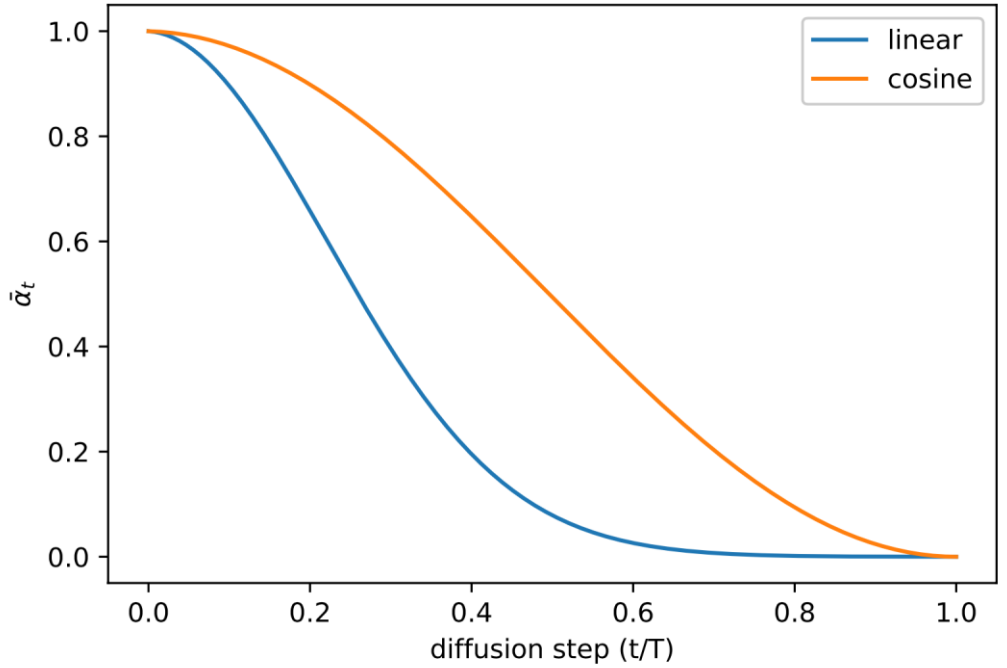
Parametrization of noise variance $\boldsymbol{\beta_t}$

The forward variances in Ho et al. (2020) are set to be linearly increasing constants from $\beta_1 = 10^{-4}$ to $\beta_T = 0.02$ (on $[-1, +1]$ normalized images (high quality images, but not yet competitive as negative log likelihood)

Nichol & Dhariwal (2021) proposed to use a cosine-based schedule

$$\beta_t = \text{clip}(1 - \frac{\bar{\alpha}_t}{\bar{\alpha}_{t-1}}, 0.999) \quad \bar{\alpha}_t = \frac{f(t)}{f(0)} \quad \text{where } f(t) = \cos\left(\frac{t/T + s}{1+s} \cdot \right.$$

where small offset s prevents β_t from being too small when close to 0.



Parametrization of reverse process variance $\boldsymbol{\Sigma_\theta}$

Ho et al. (2020) fix $\Sigma_\theta(x_t, t) = \sigma_t^2 I$ setting σ_t^2 equal to β_t or $\tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}$ as learning reverse process variance was unstable and not effective.

Nichol & Dhariwal (2021) proposed to learn $\Sigma_\theta(x_t, t)$ interpolating β_t and $\tilde{\beta}_t$ with a learned mixing vector v

$$\boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t) = \exp(\mathbf{v} \log \beta_t + (1 - \mathbf{v}) \log \tilde{\beta}_t)$$

To learn this they used a hybrid loss with a variational lower bound

$$L_{\text{hybrid}} = L_{\text{simple}} + \lambda L_{\text{VLB}} \text{ where } \lambda = 0.001$$

which turns anyway hard to optimize due to likely to noisy gradients …

Model	ImageNet	CIFAR
Glow (Kingma & Dhariwal, 2018)	3.81	3.35
Flow++ (Ho et al., 2019)	3.69	3.08
PixelCNN (van den Oord et al., 2016c)	3.57	3.14
SPN (Menick & Kalchbrenner, 2018)	3.52	-
NVAE (Vahdat & Kautz, 2020)	-	2.91
Very Deep VAE (Child, 2020)	3.52	2.87
PixelSNAIL (Chen et al., 2018)	3.52	2.85
Image Transformer (Parmar et al., 2018)	3.48	2.90
Sparse Transformer (Child et al., 2019)	3.44	**2.80**
Routing Transformer (Roy et al., 2020)	**3.43**	-
DDPM (Ho et al., 2020)	3.77	3.70
DDPM (cont flow) (Song et al., 2020b)	-	2.99
Improved DDPM (ours)	**3.53**	**2.94**

POLITECNICO
MILANO 1863

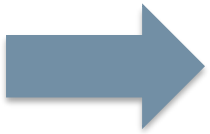
Kemal Erdem, (Nov 2023). "Step by Step visual introduction to Diffusion Models.".
https://erdem.pl/2023/11/step-by-step-visual-introduction-to-diffusion-models

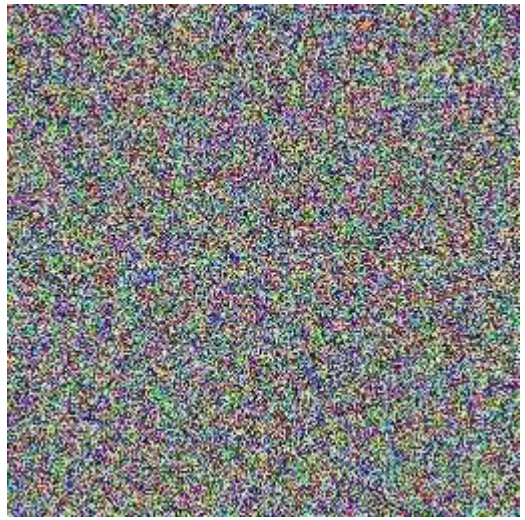
Step by Step Diffusion Models

Matteo Matteucci, PhD (matteo.matteucci@polimi.it)
Artificial Intelligence and Robotics Laboratory
Politecnico di Milano

AIRLAB
ARTIFICIAL INTELLIGENCE AND ROBOTICS LAB

Forward Diffusion Process



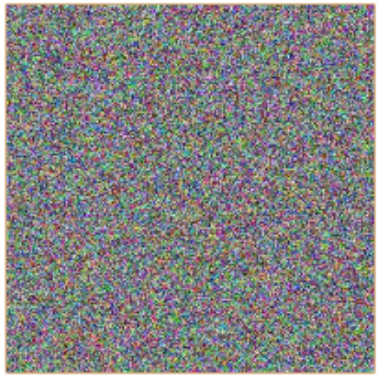


Forward Diffusion Process

t=0

Forward Diffusion Process

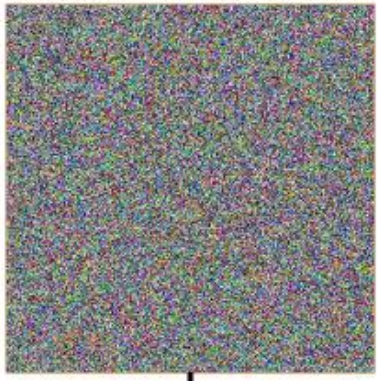
t=0



Generate noise for current step **t**

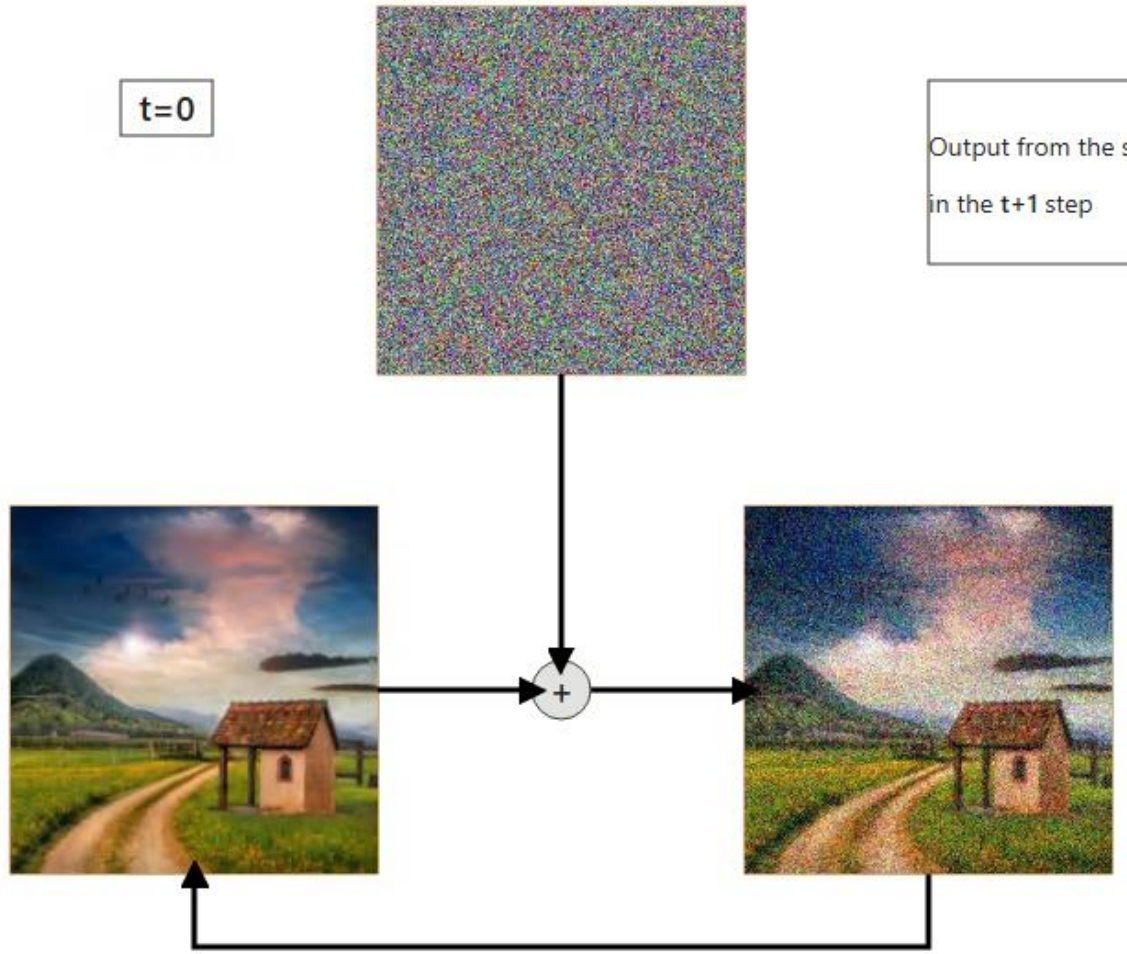
Forward Diffusion Process

t=0



Add noise to the input image using linear scheduler

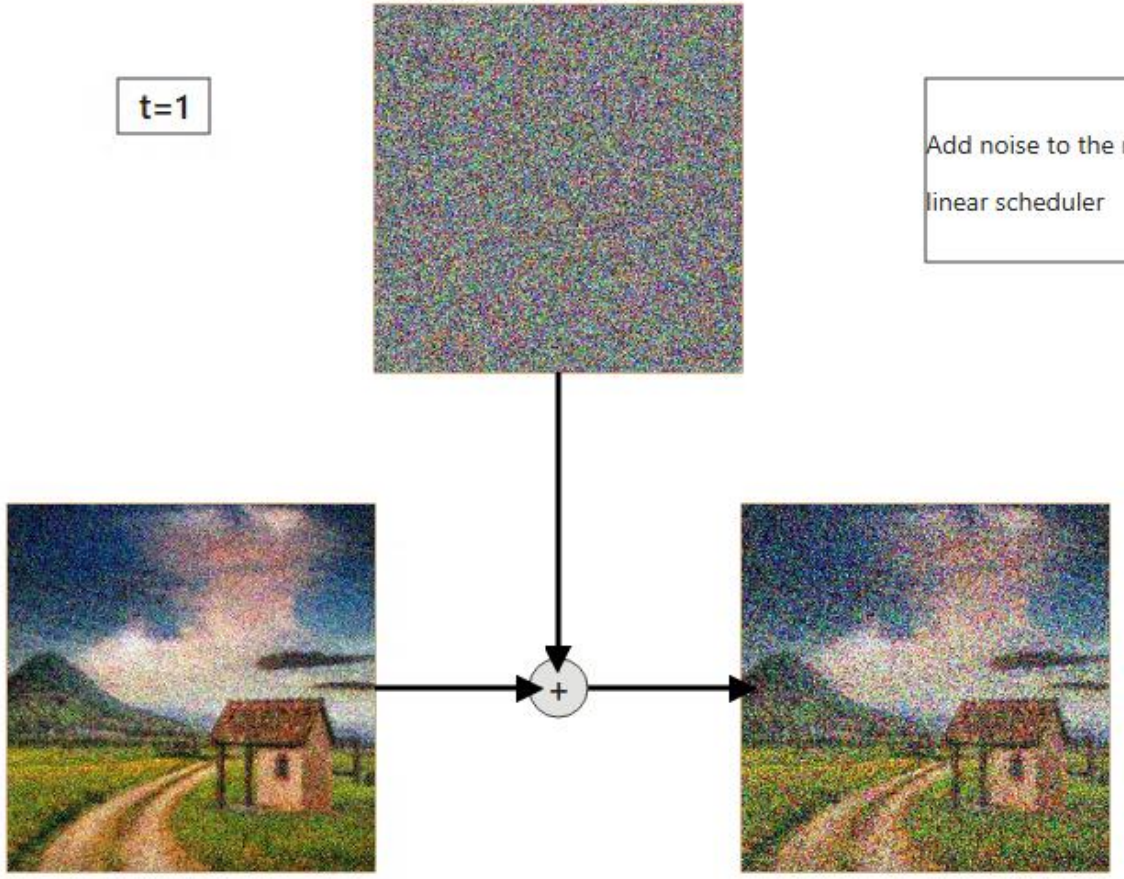
Forward Diffusion Process



Forward Diffusion Process

t=1

Forward Diffusion Process

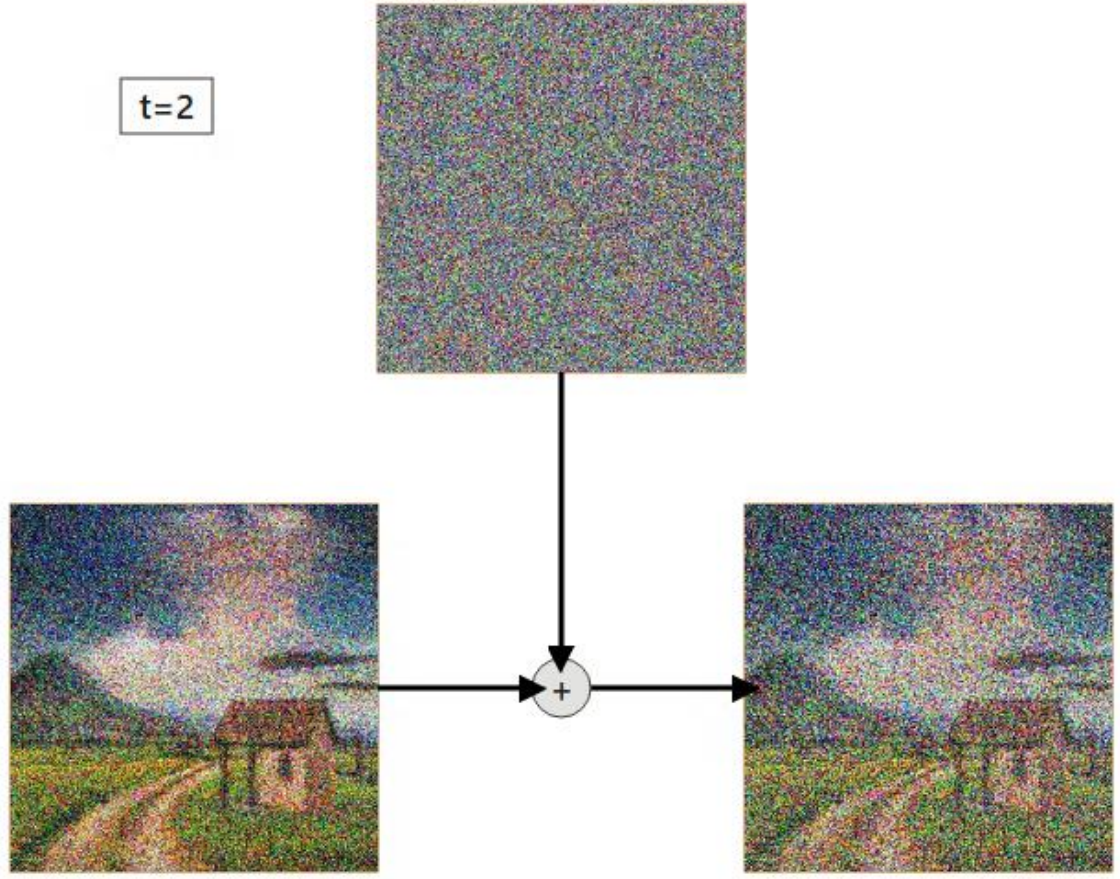


Forward Diffusion Process

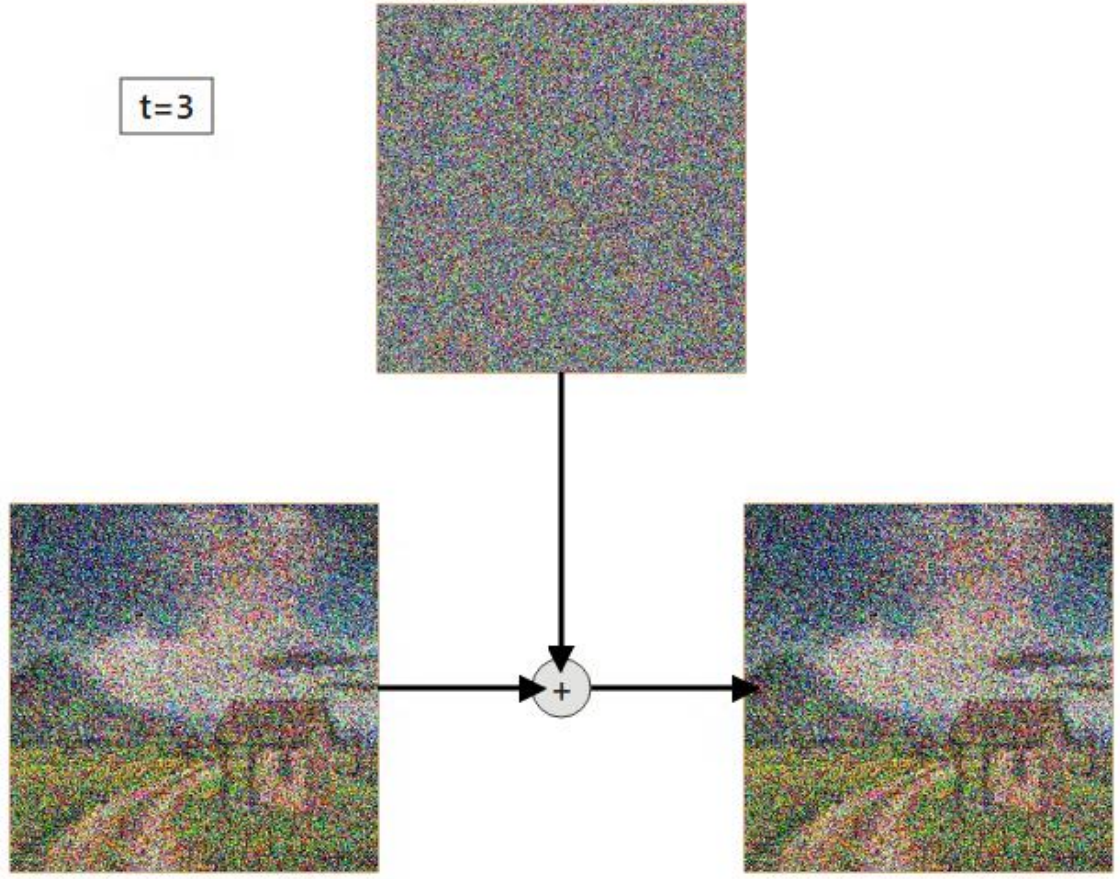
t=2

Output from the step **t** becomes input again and the whole process repeats

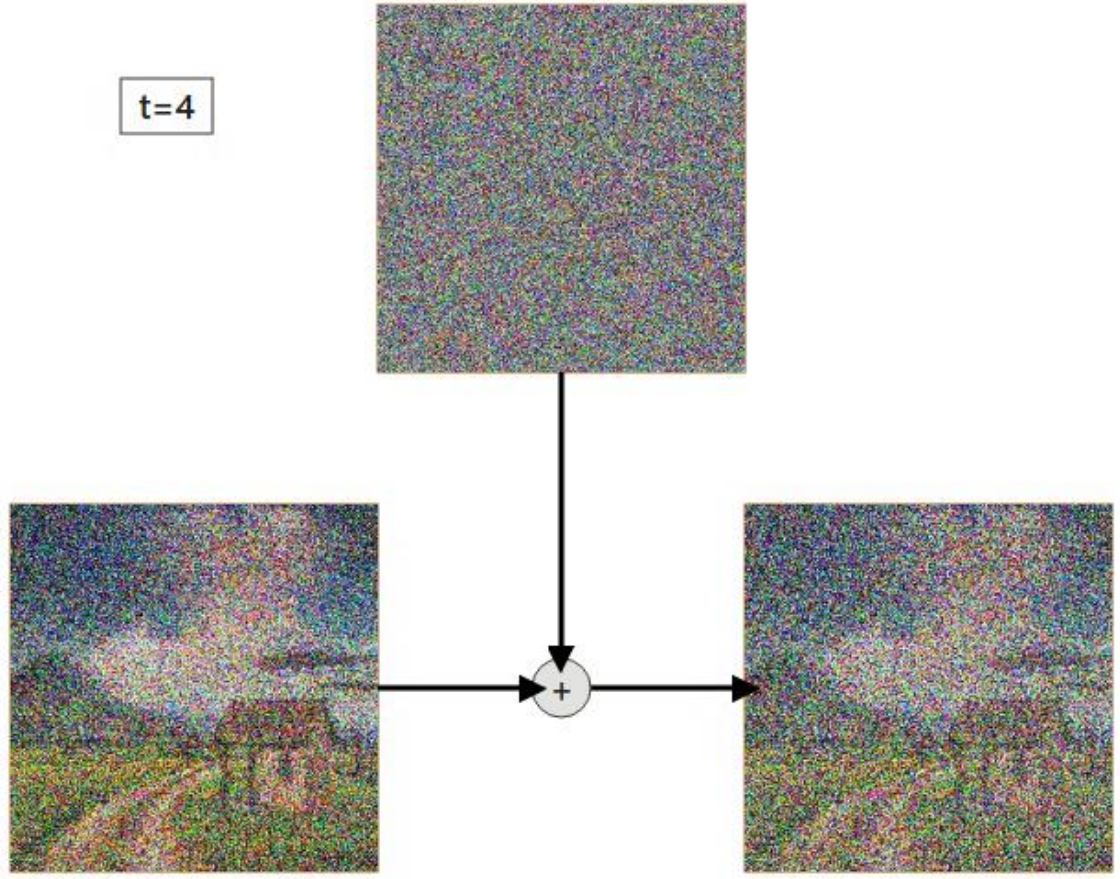
Forward Diffusion Process



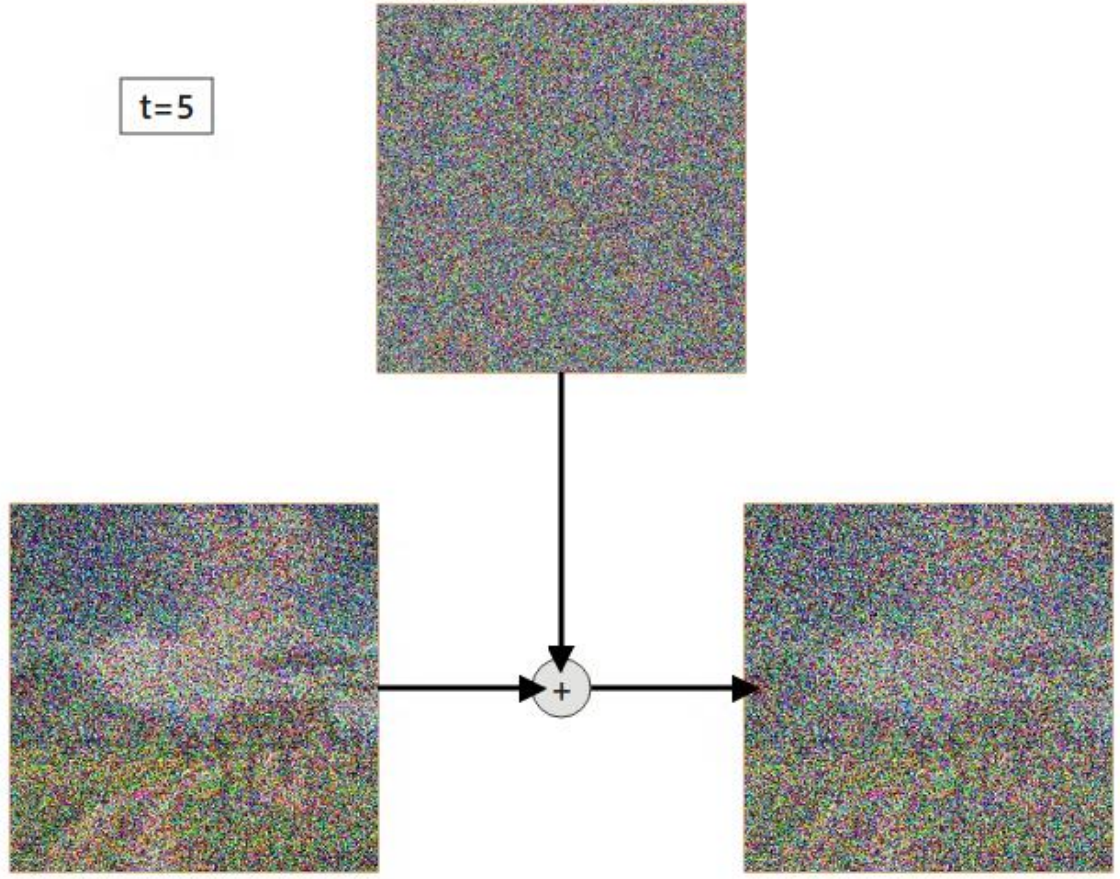
Forward Diffusion Process



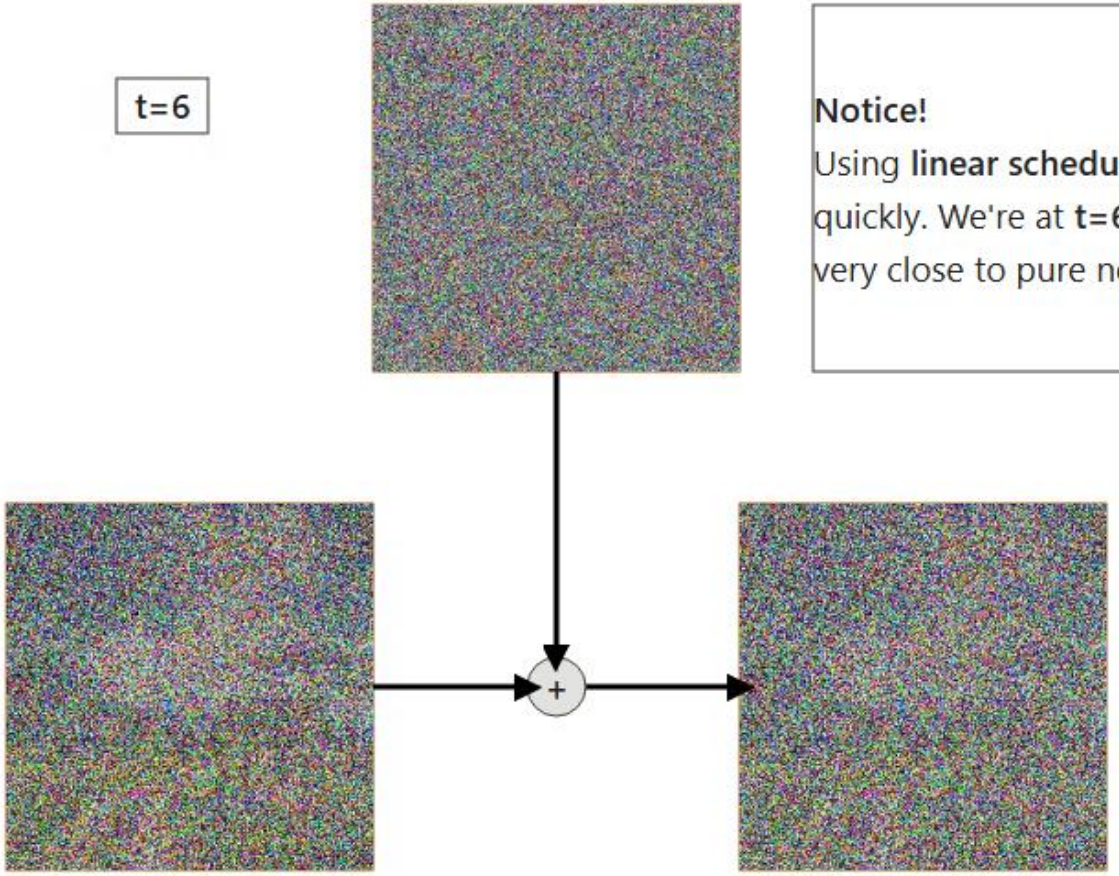
Forward Diffusion Process



Forward Diffusion Process

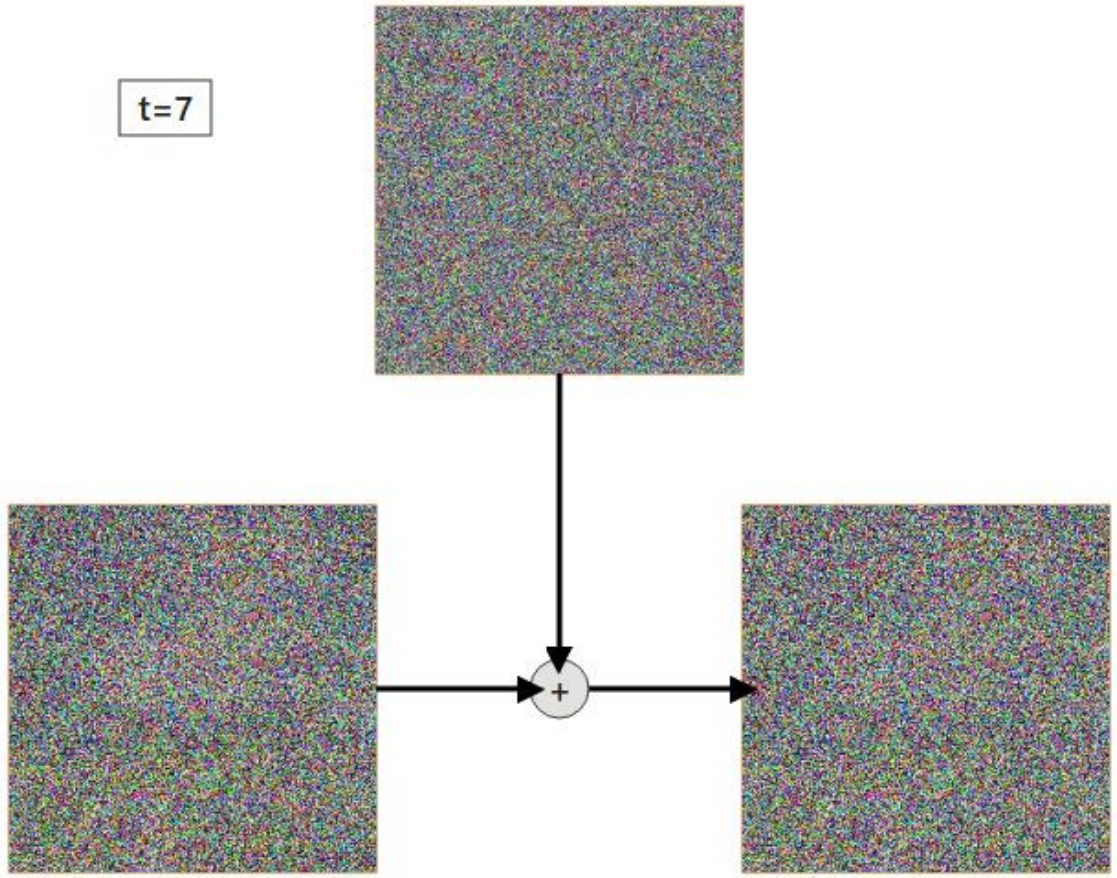


Forward Diffusion Process

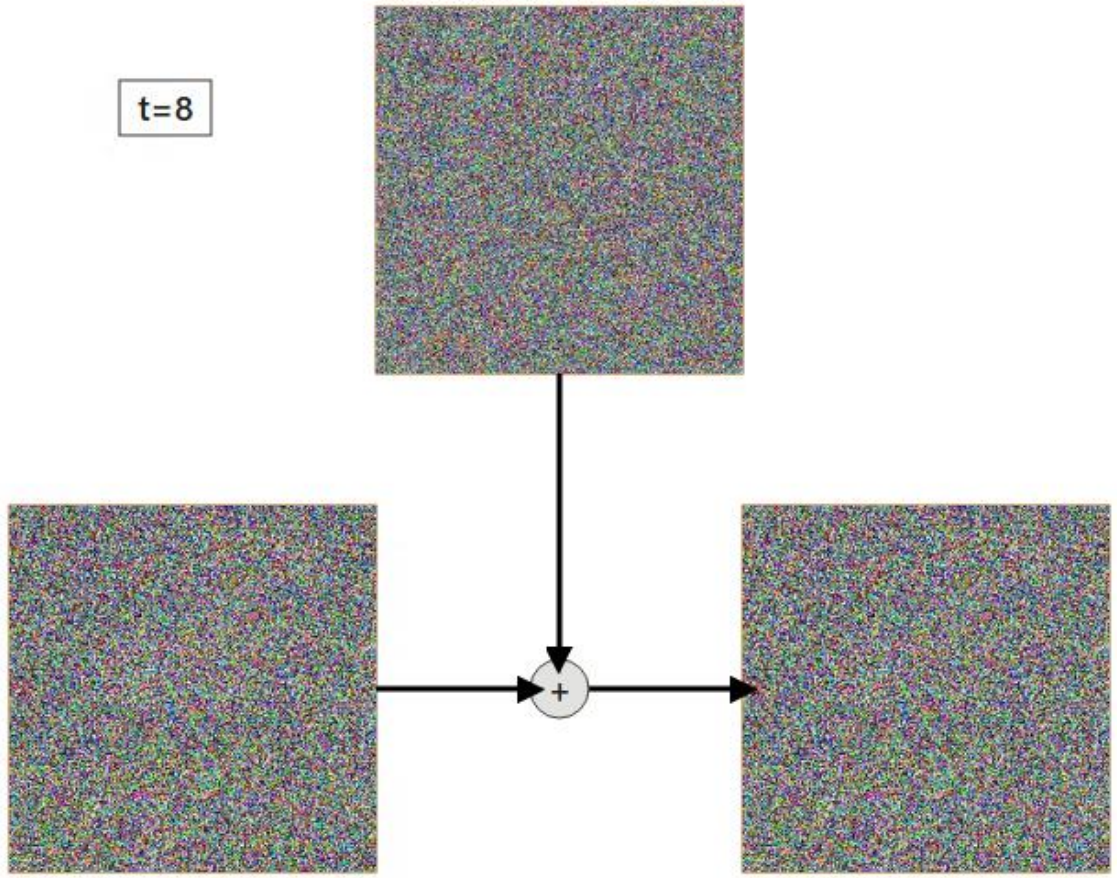


Notice!
Using **linear scheduler**, the information is lost pretty quickly. We're at **t=6** and the output image is already very close to pure noise.

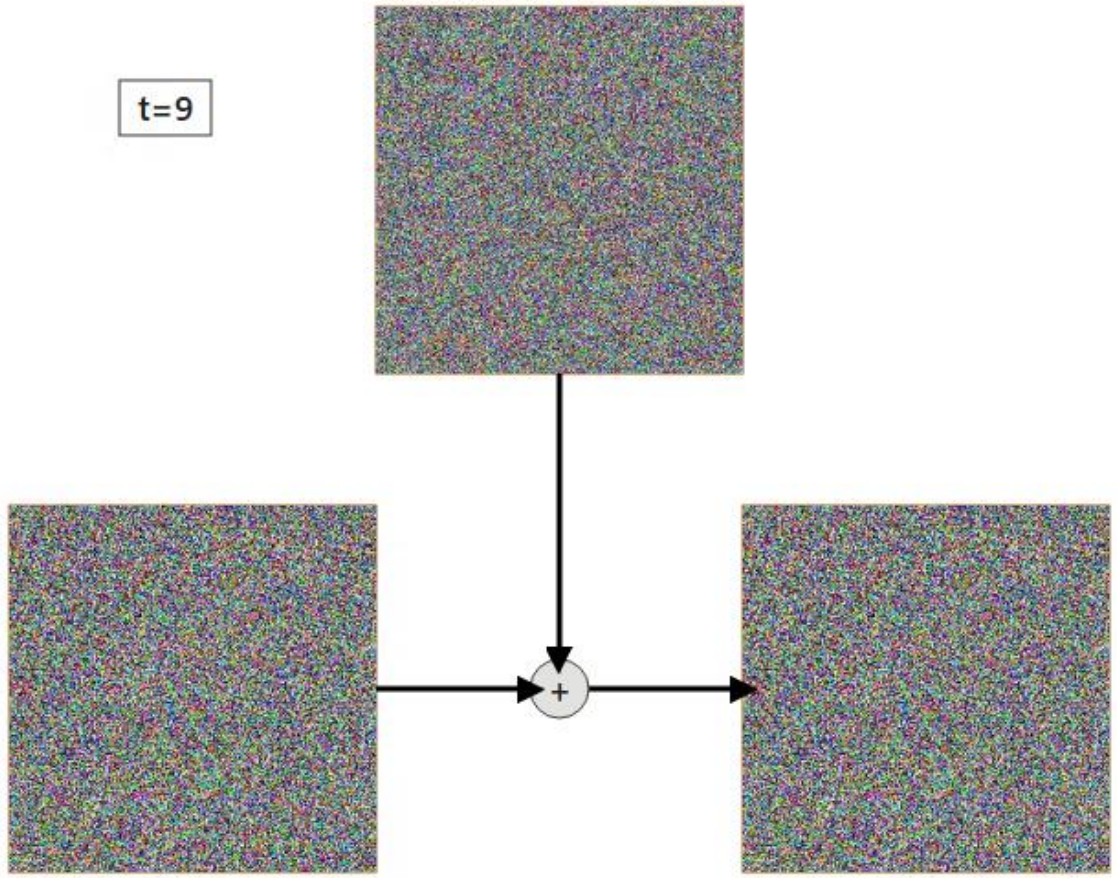
Forward Diffusion Process



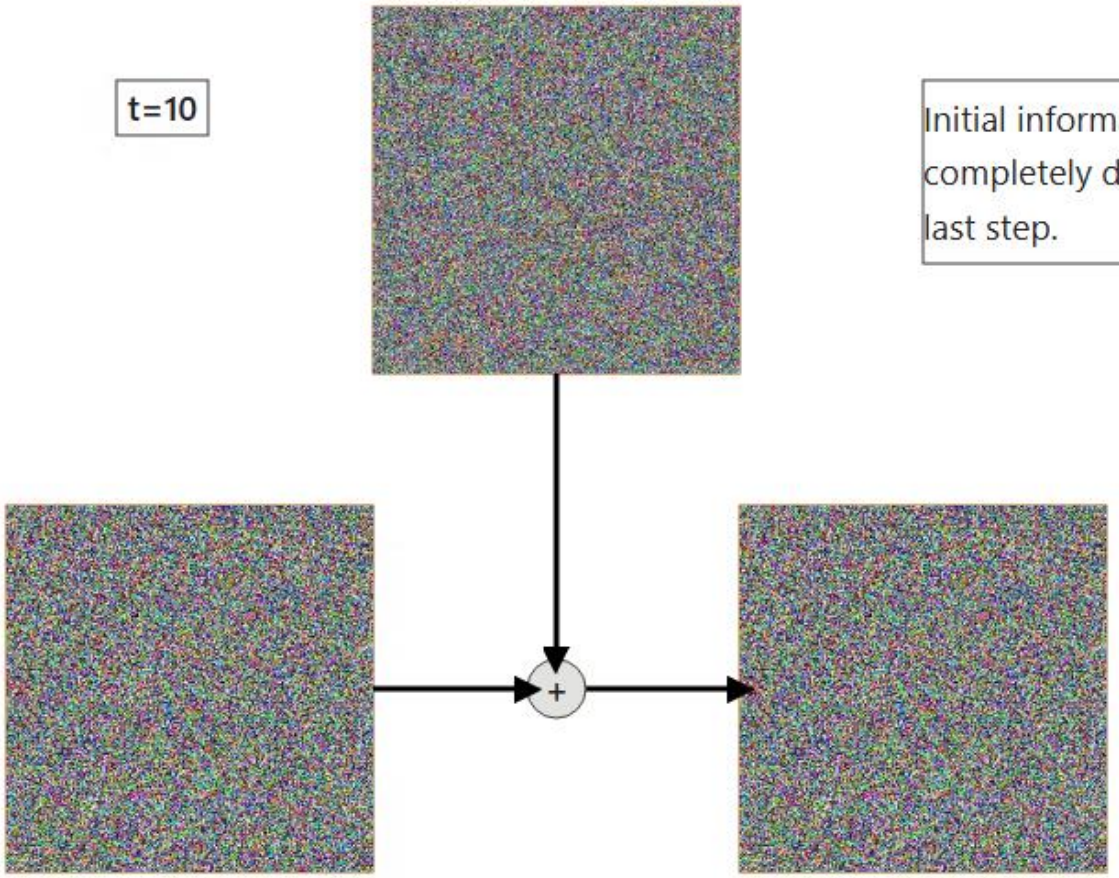
Forward Diffusion Process



Forward Diffusion Process



Forward Diffusion Process



Initial information is completely destroyed at the last step.

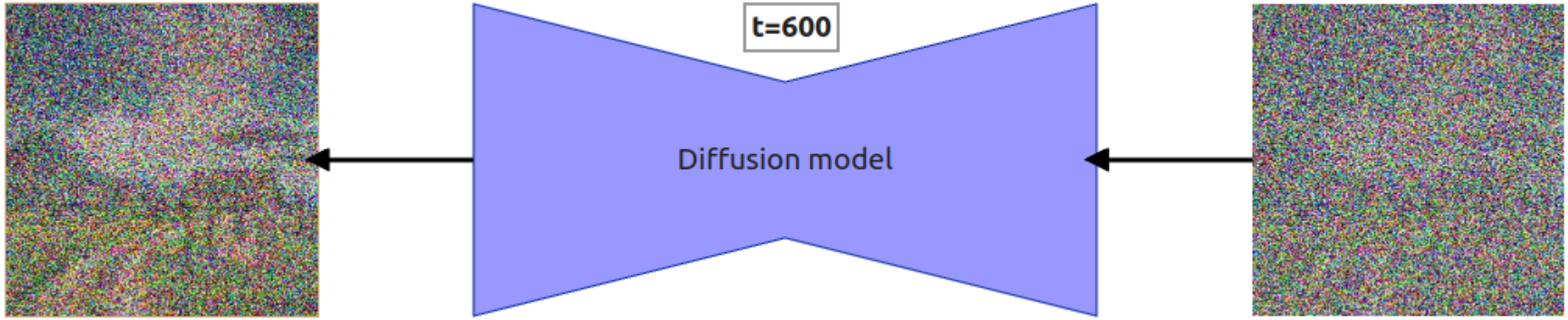
Linear vs Cosine Scheduling

Linear scheduling

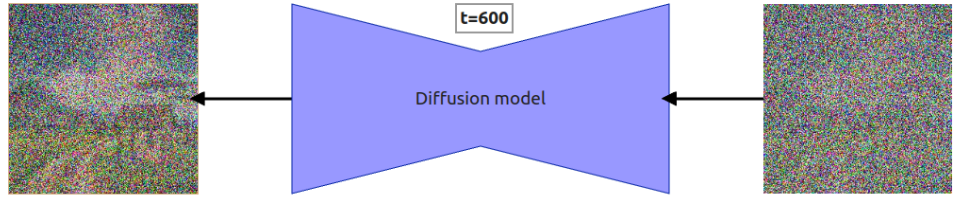
Cosine Scheduling

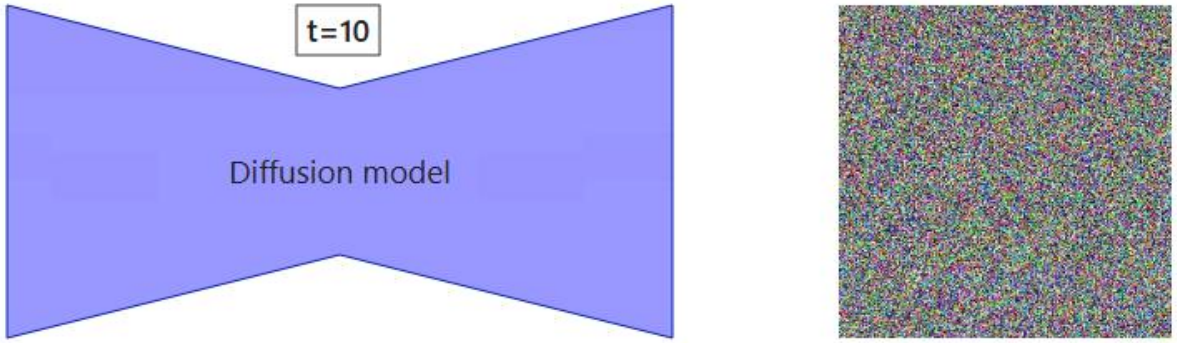
Reverse Diffusion Process

The model has been trained to predict the entire noise to be removed at a given timestep, e.g., 600 in this picture, to recover the original image.

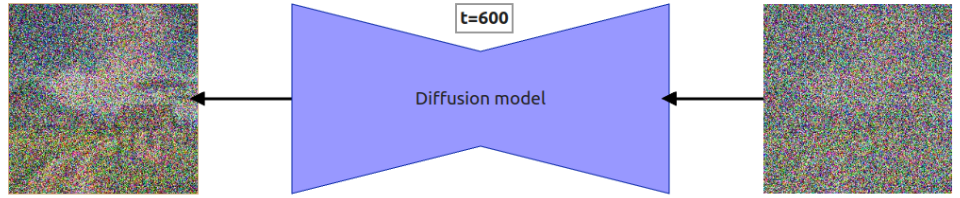


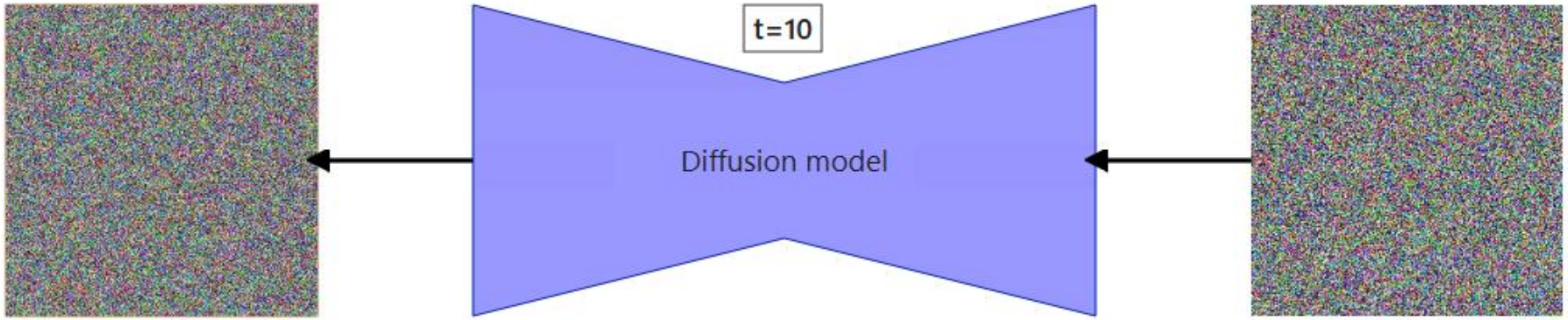
Reverse Diffusion Process





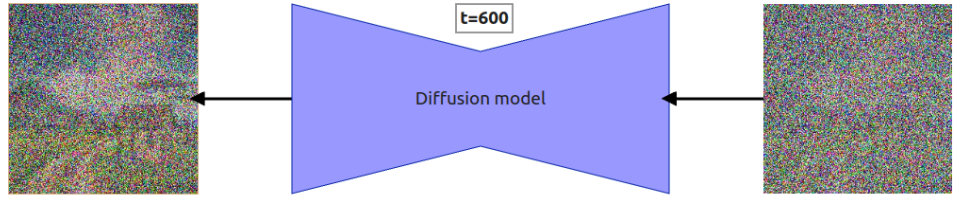
Reverse Diffusion Process

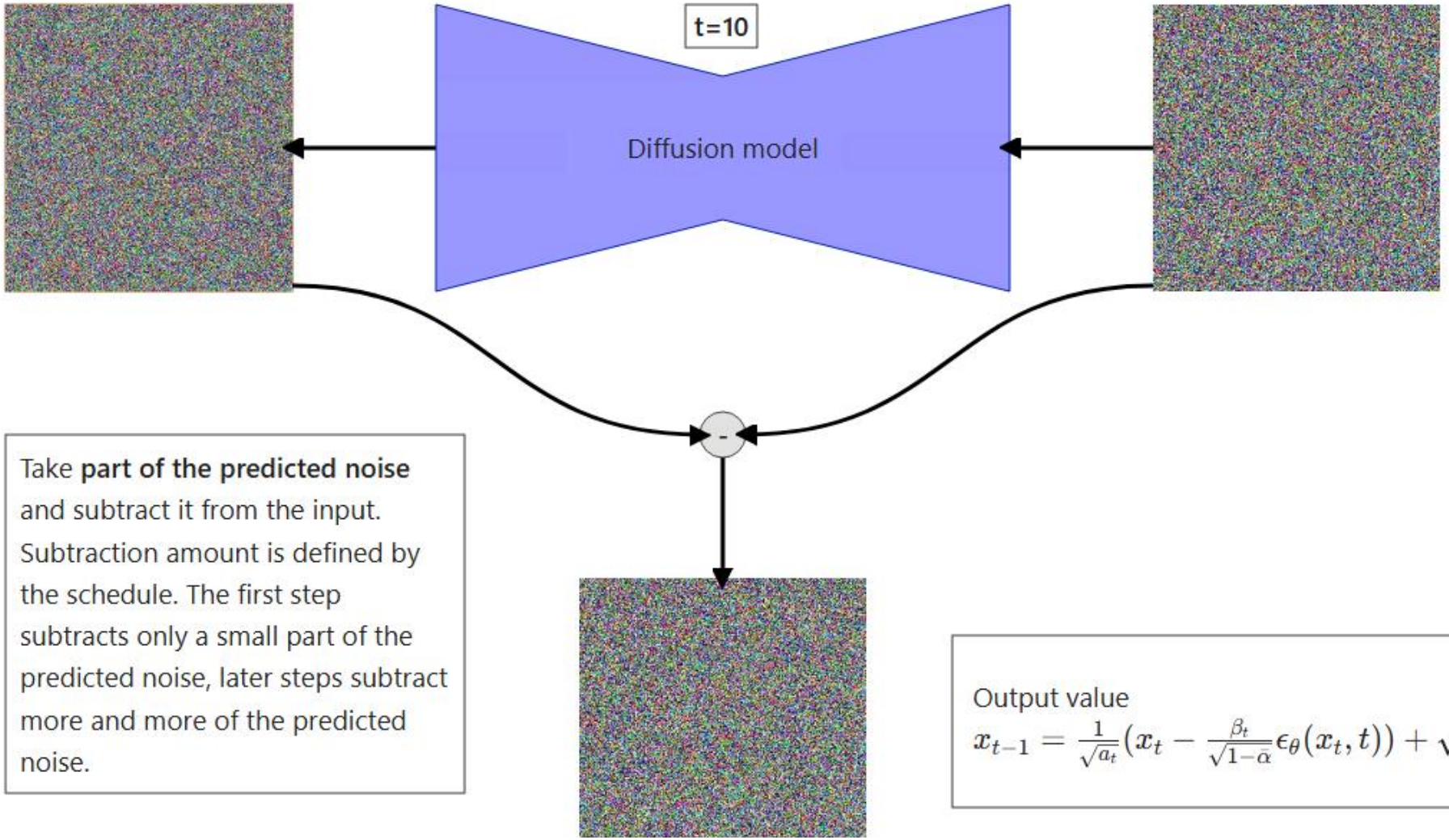




Notice!
Output of the model is an **entire noise** that the model predicted to be removed from the input. This noise is later scaled based on the schedule to be subtracted from the input.

Reverse Diffusion Process



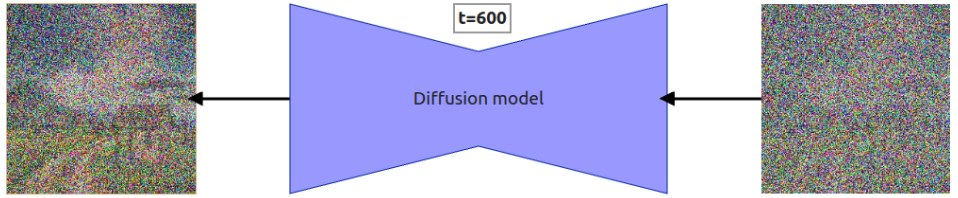


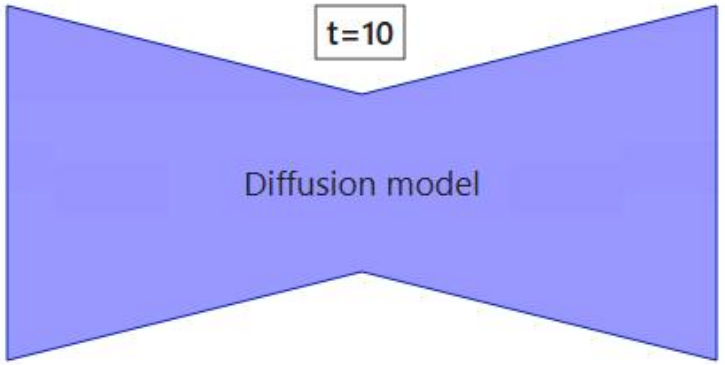
Take **part of the predicted noise** and subtract it from the input. Subtraction amount is defined by the schedule. The first step subtracts only a small part of the predicted noise, later steps subtract more and more of the predicted noise.

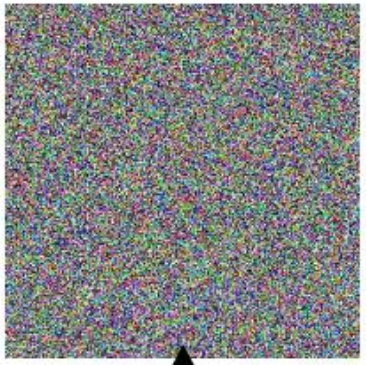
Output value

$$x_{t-1} = \frac{1}{\sqrt{a_t}}(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}}}\epsilon_\theta(x_t, t)) + \sqrt{\beta_t}\epsilon$$

Reverse Diffusion Process





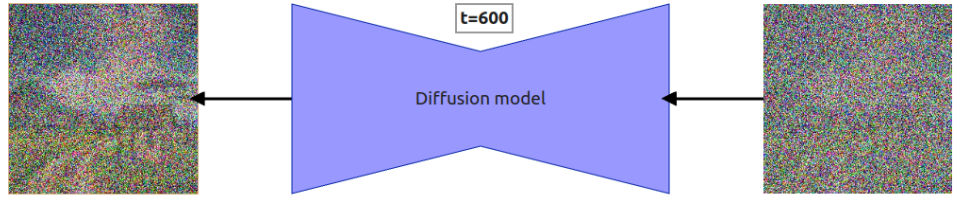


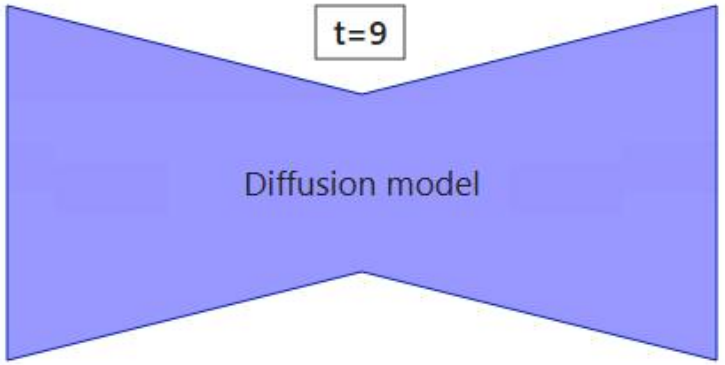
Output from the **t** timestep becomes an input in **t-1**.

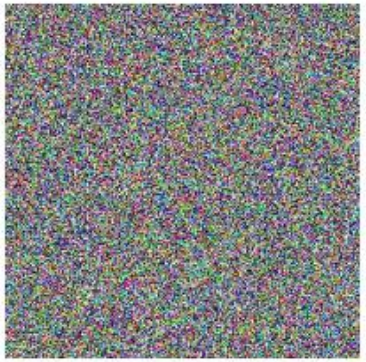
Output value

$$x_{t-1} = \frac{1}{\sqrt{a_t}}(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}}}\epsilon_\theta(x_t, t)) + \sqrt{\beta_t}\epsilon$$

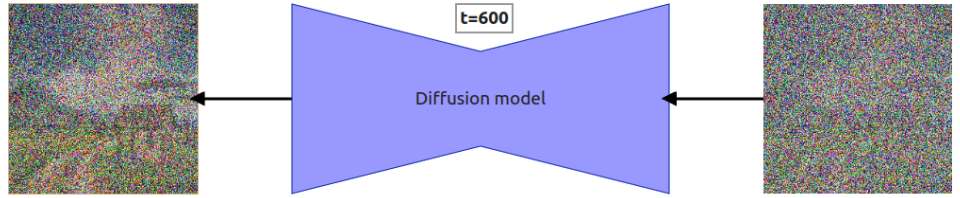
Reverse Diffusion Process

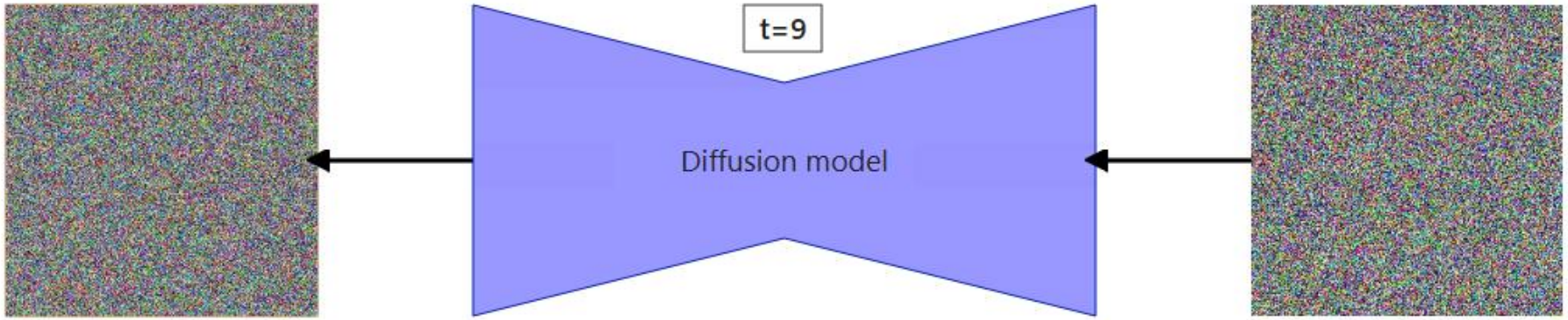






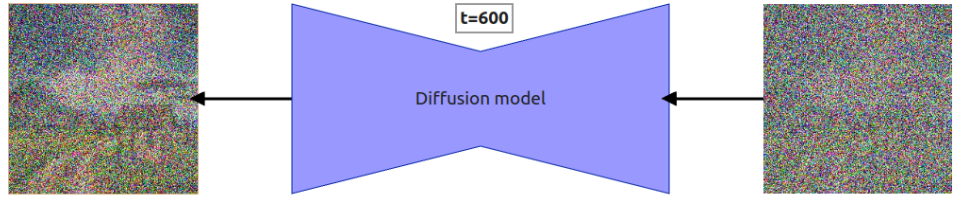
Reverse Diffusion Process

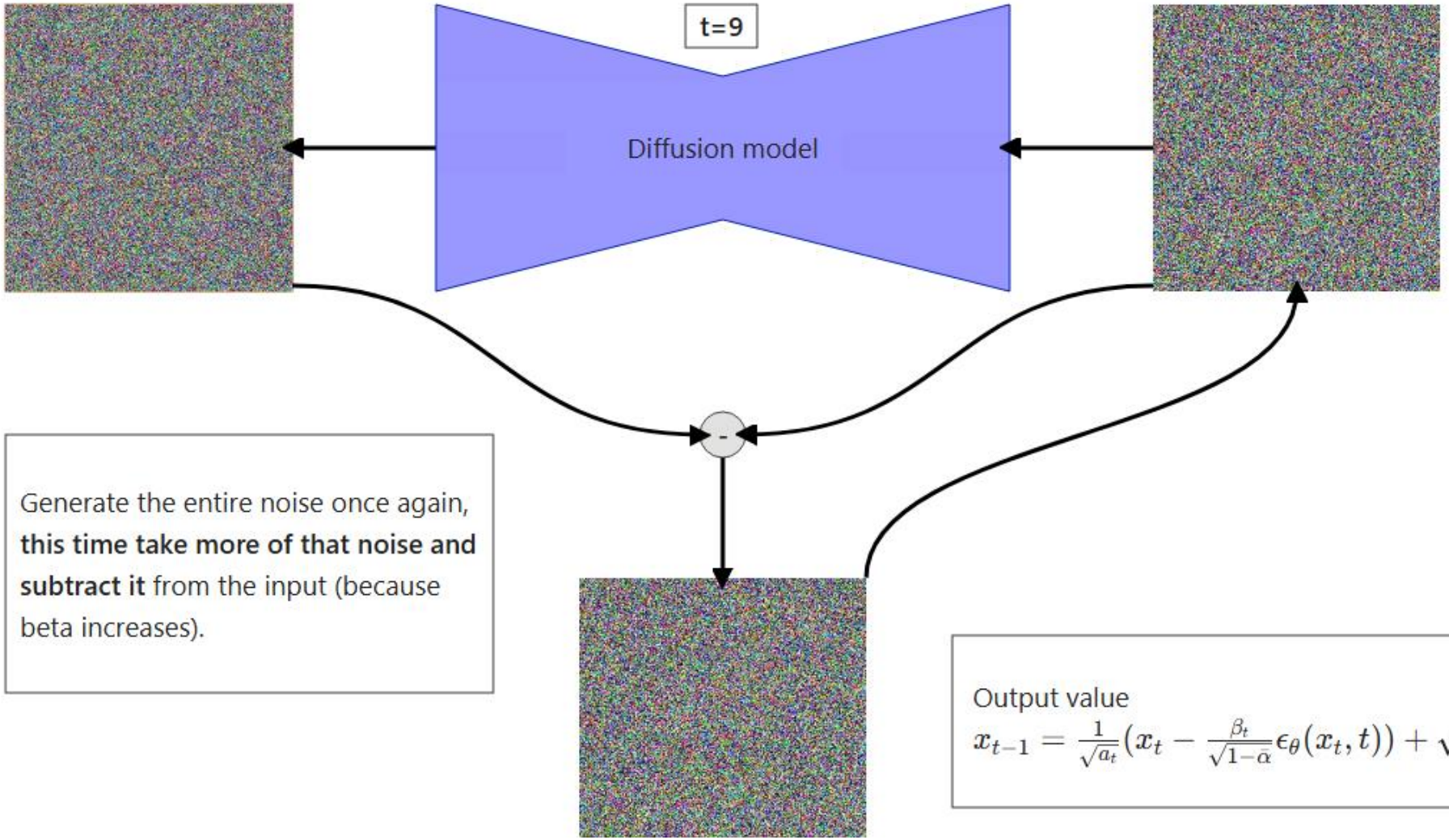




Generate the entire noise once again, **this time take more of that noise and subtract it** from the input (because beta increases).

Reverse Diffusion Process



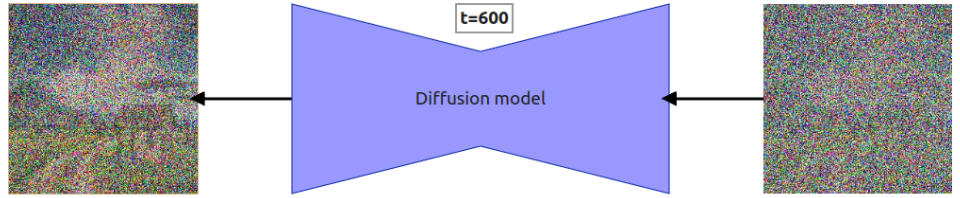


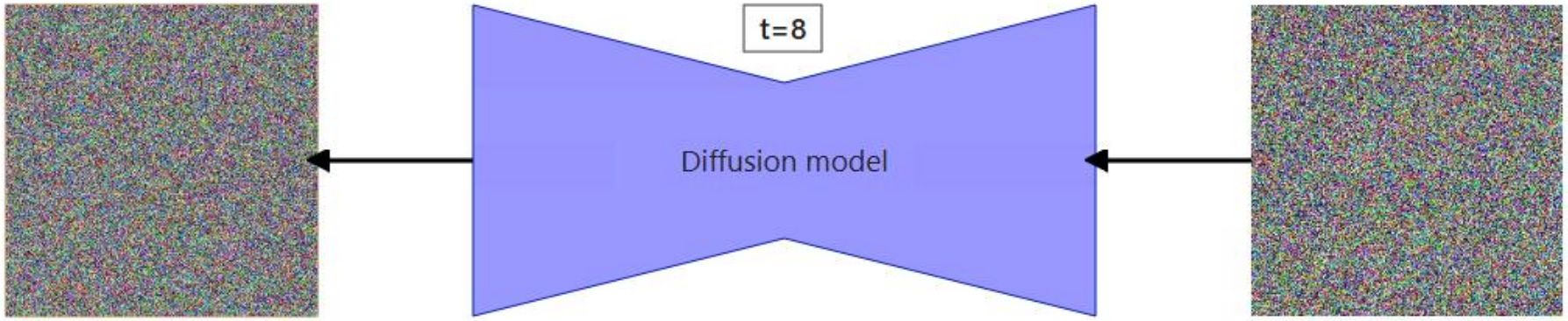
Generate the entire noise once again, **this time take more of that noise and subtract it** from the input (because beta increases).

Output value

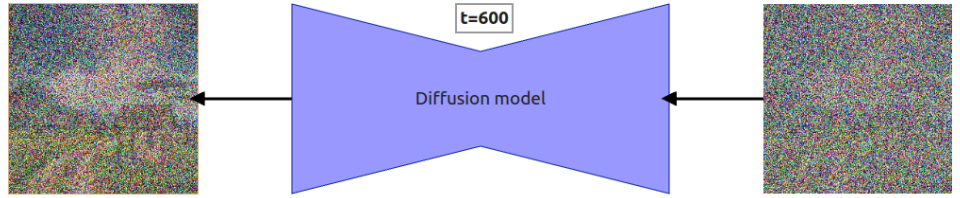
$$x_{t-1} = \frac{1}{\sqrt{a_t}}(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}}}\epsilon_\theta(x_t, t)) + \sqrt{\beta_t}\epsilon$$

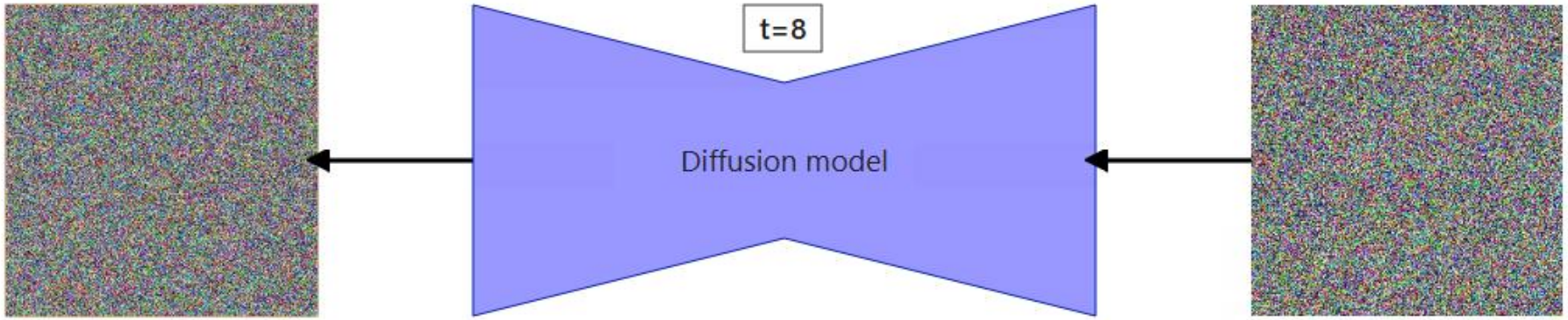
Reverse Diffusion Process





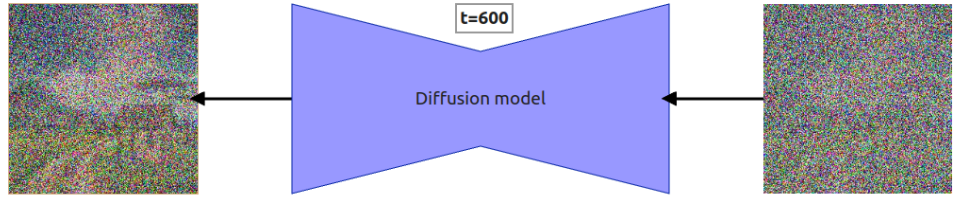
Reverse Diffusion Process

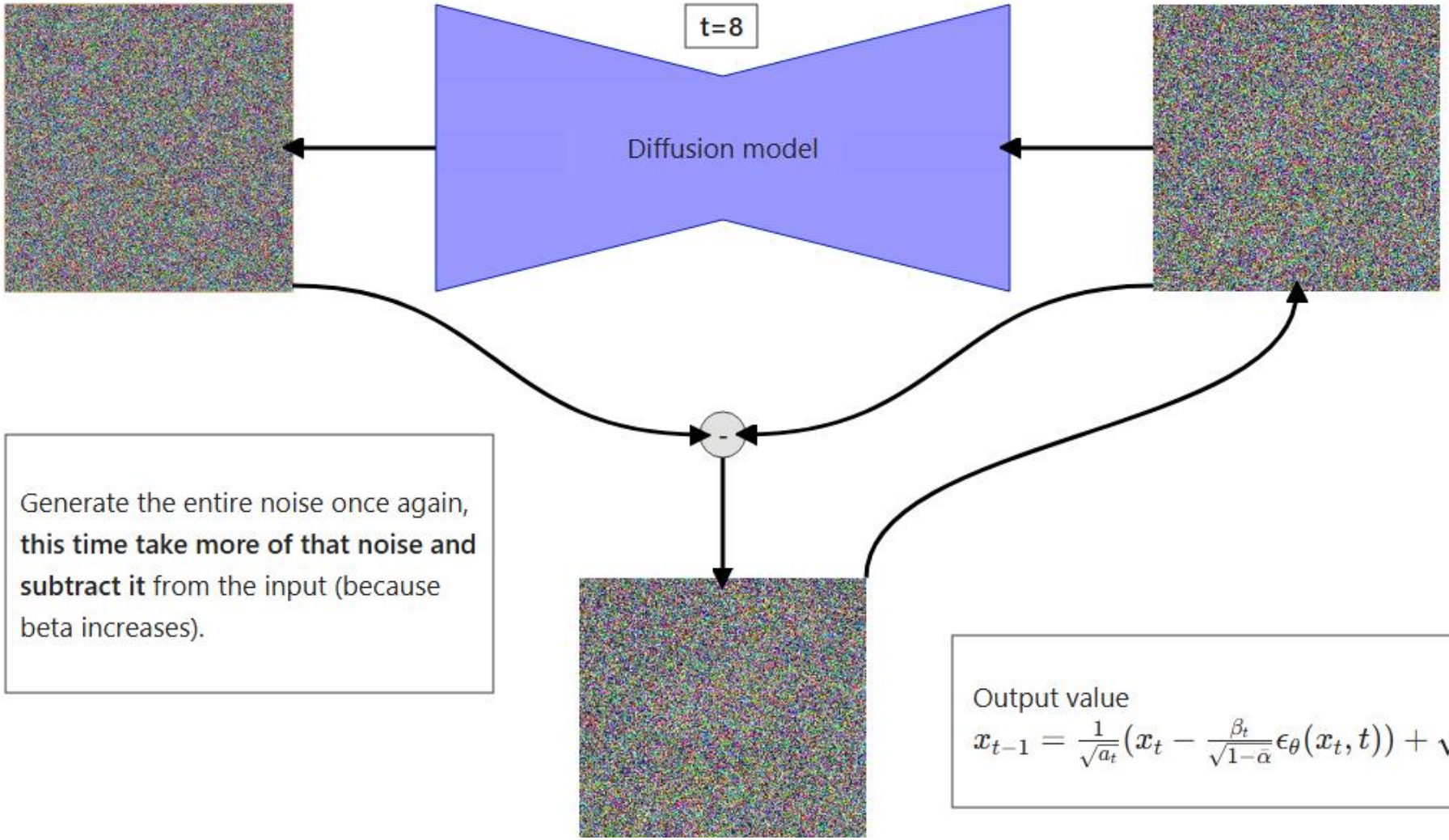




Generate the entire noise once again, **this time take more of that noise and subtract it** from the input (because beta increases).

Reverse Diffusion Process



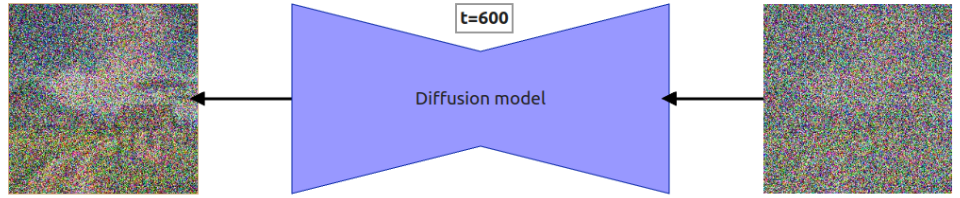


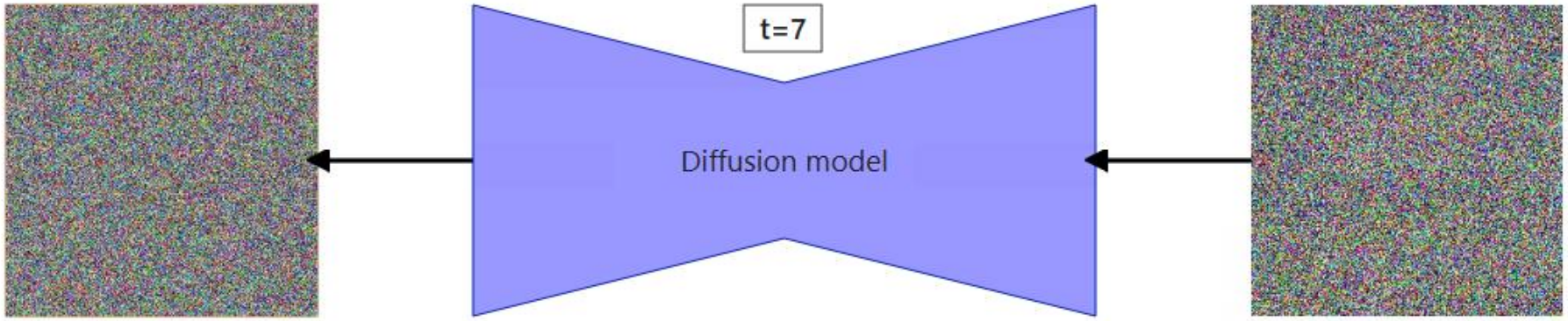
Generate the entire noise once again, **this time take more of that noise and subtract it** from the input (because beta increases).

Output value

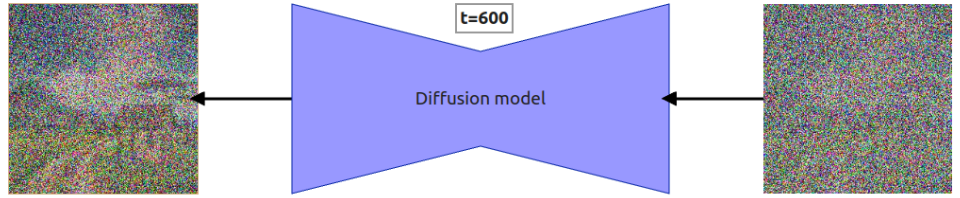
$$x_{t-1} = \frac{1}{\sqrt{a_t}}(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}}}\epsilon_\theta(x_t, t)) + \sqrt{\beta_t}\epsilon$$

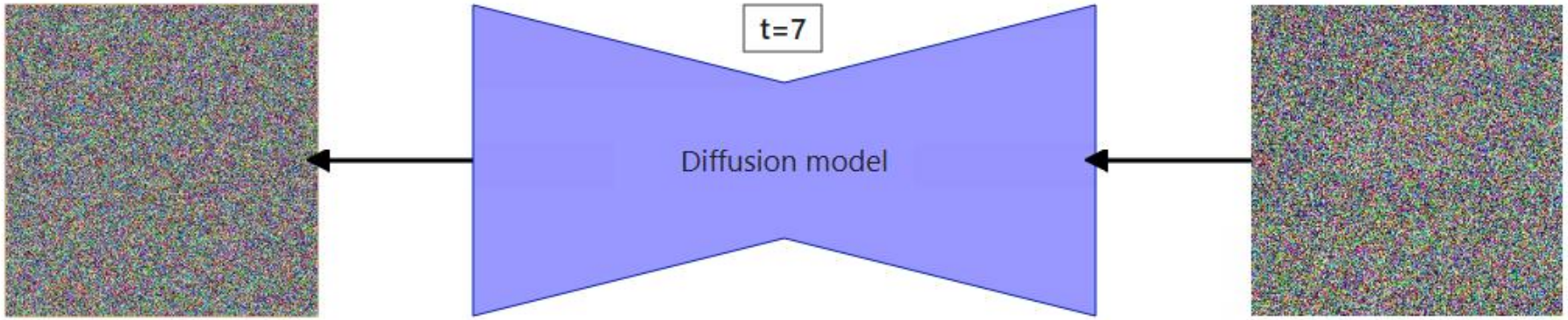
Reverse Diffusion Process





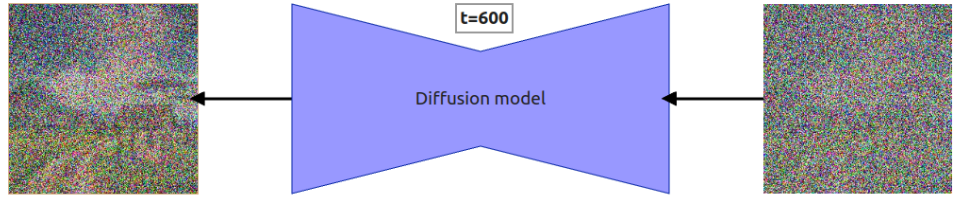
Reverse Diffusion Process

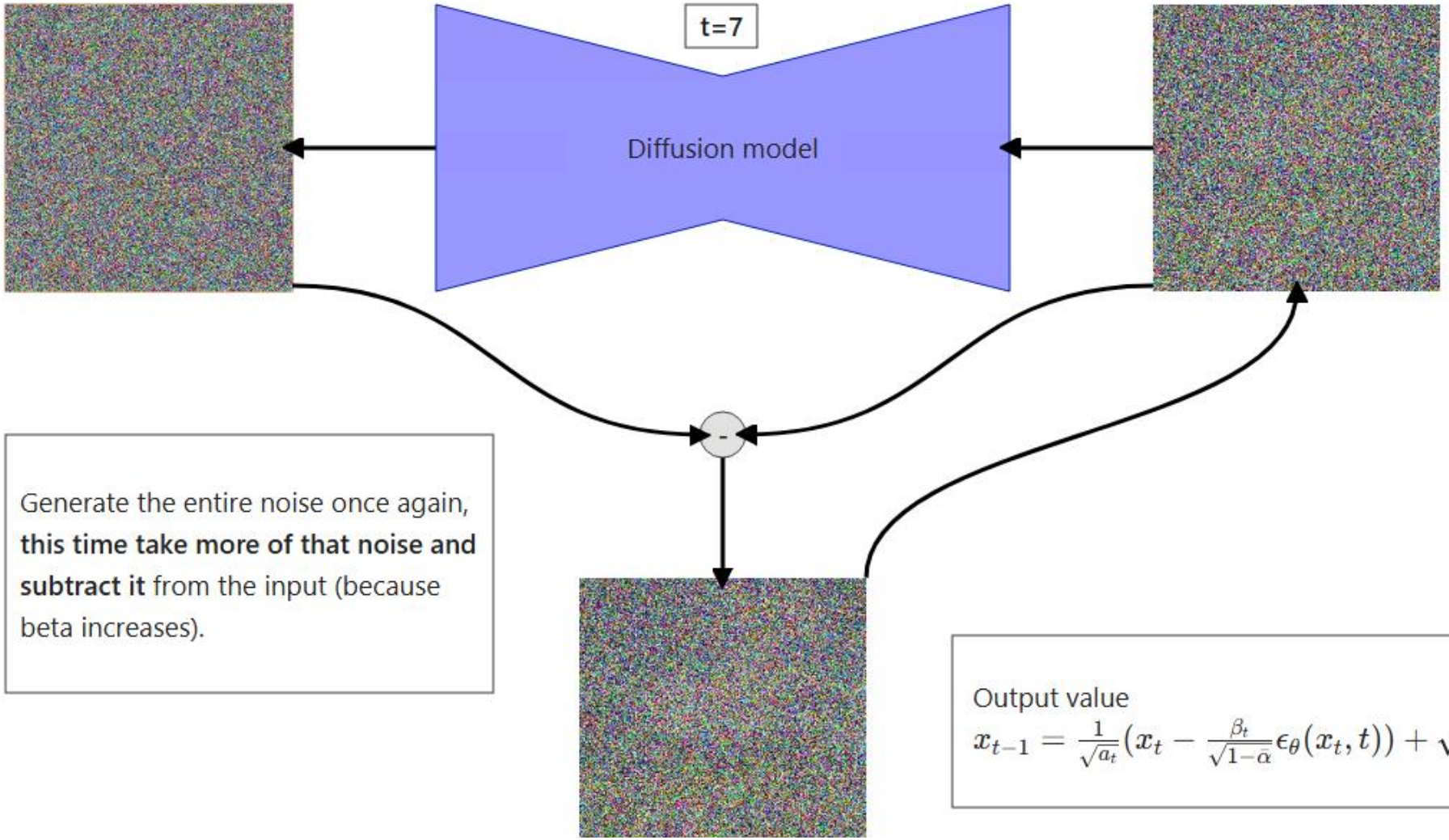




Generate the entire noise once again, **this time take more of that noise and subtract it** from the input (because beta increases).

Reverse Diffusion Process



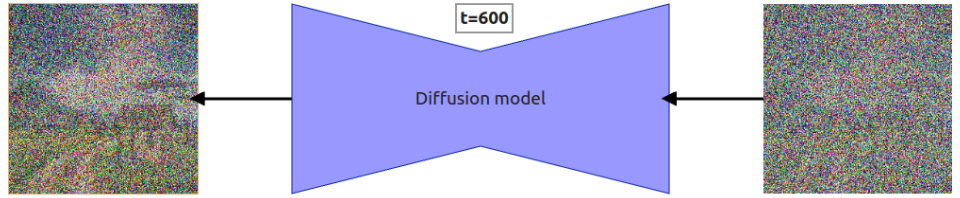


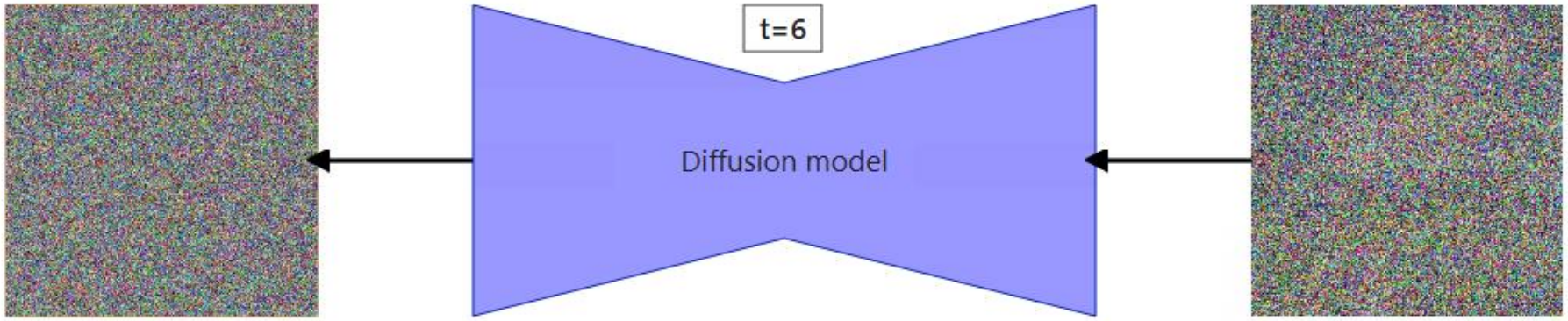
Generate the entire noise once again, **this time take more of that noise and subtract it** from the input (because beta increases).

Output value

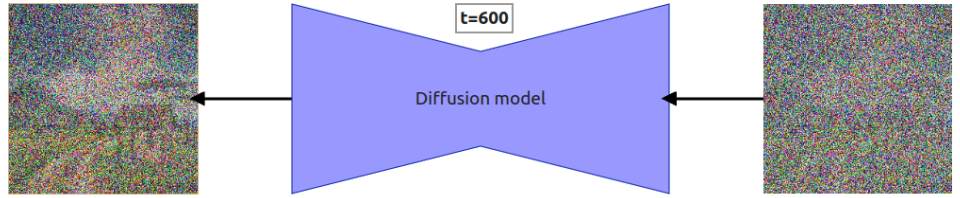
$$x_{t-1} = \frac{1}{\sqrt{a_t}}(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}}}\epsilon_\theta(x_t, t)) + \sqrt{\beta_t}\epsilon$$

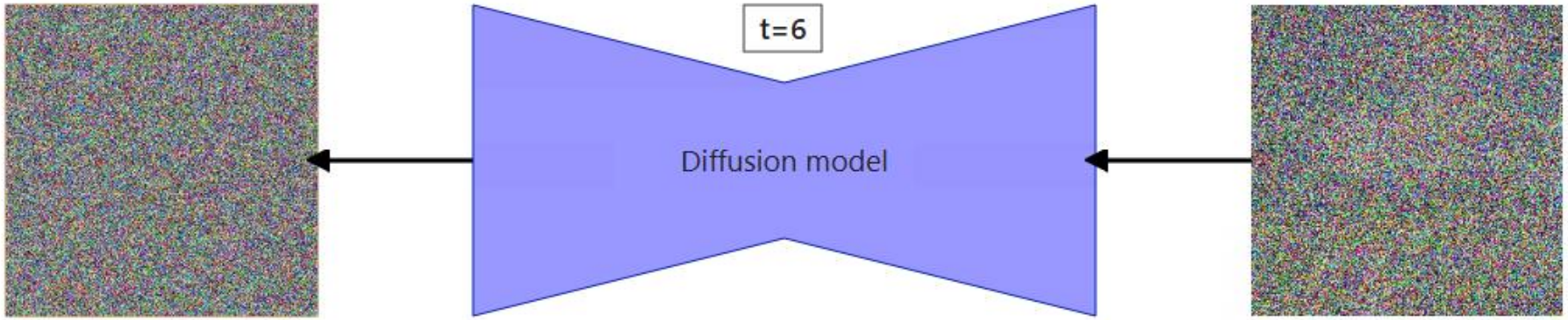
Reverse Diffusion Process





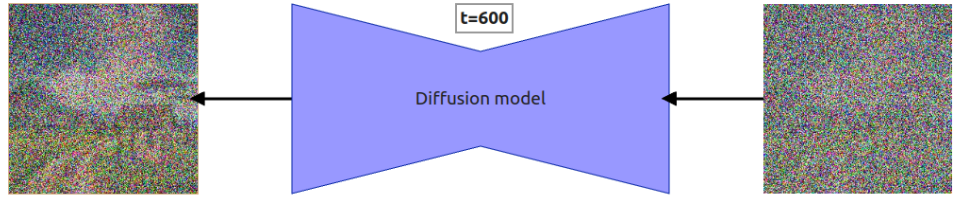
Reverse Diffusion Process

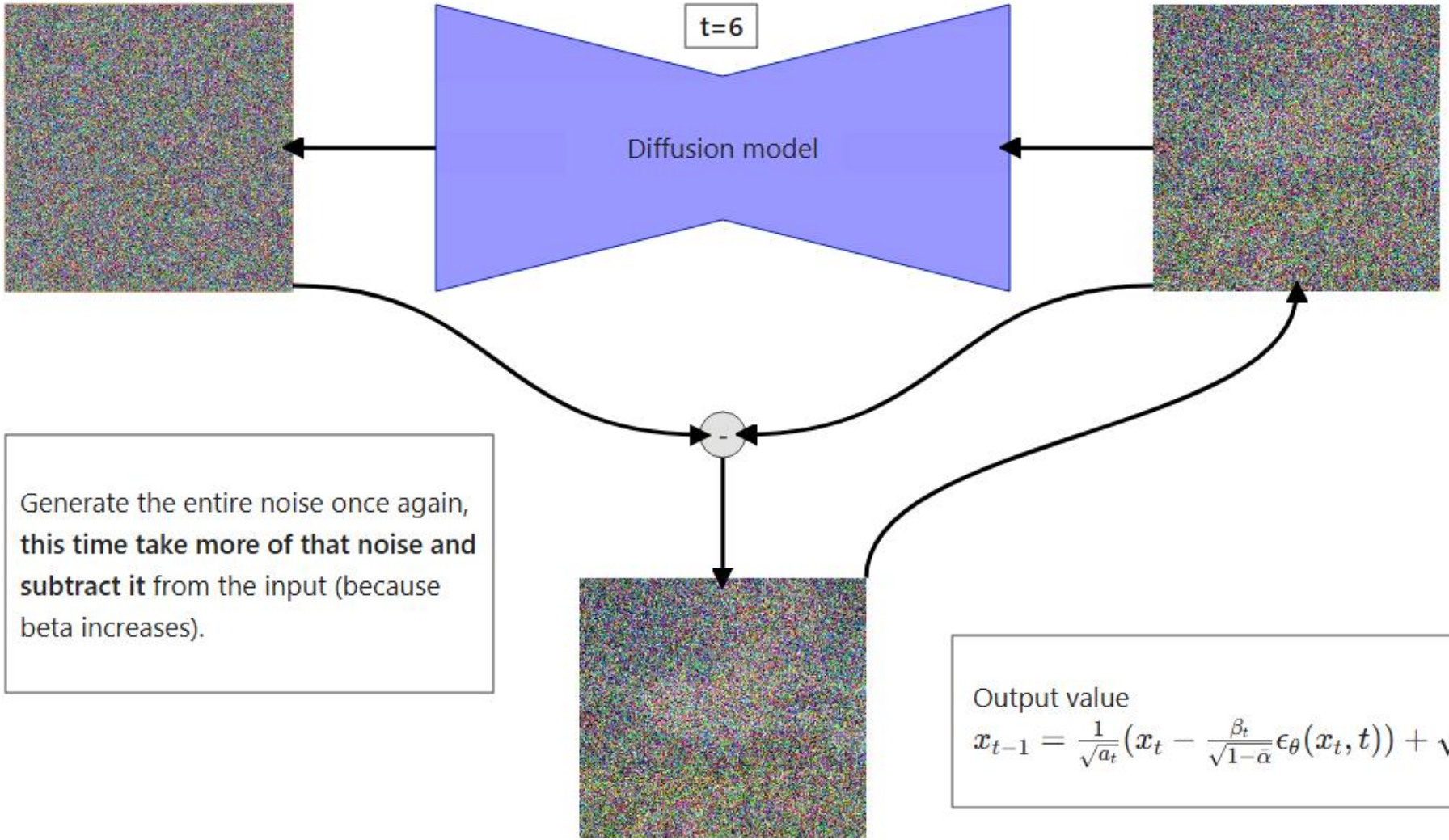




Generate the entire noise once again, **this time take more of that noise and subtract it** from the input (because beta increases).

Reverse Diffusion Process



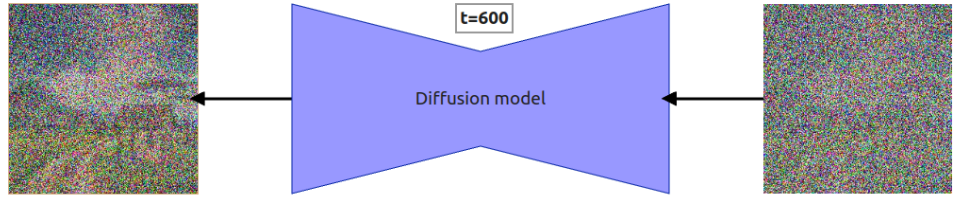


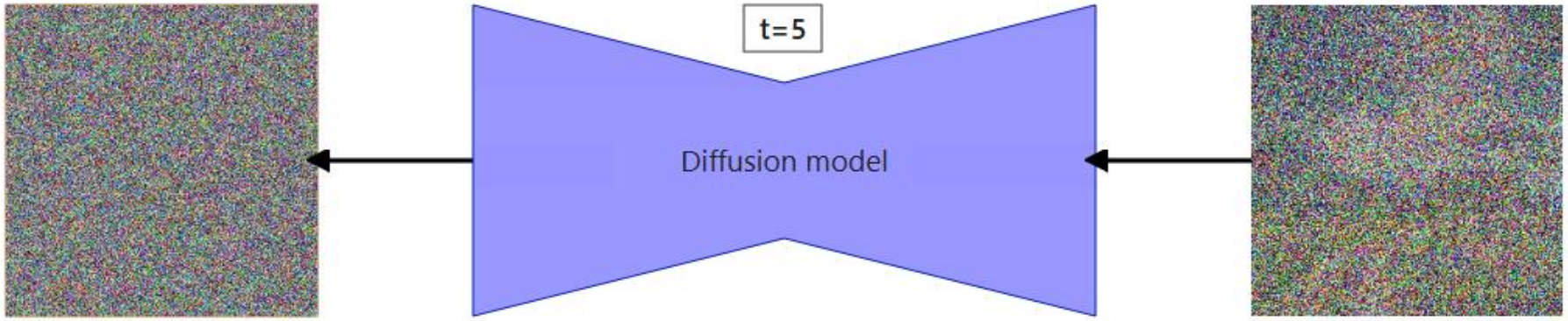
Generate the entire noise once again, **this time take more of that noise and subtract it** from the input (because beta increases).

Output value

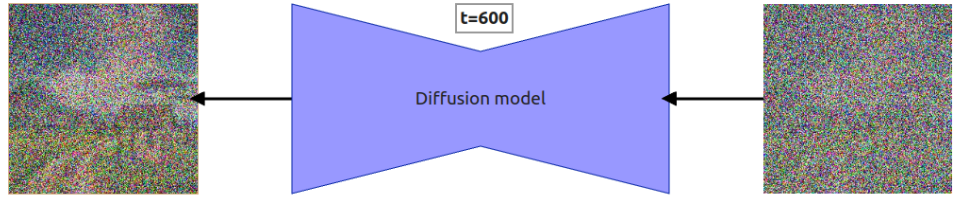
$$x_{t-1} = \frac{1}{\sqrt{a_t}}(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}}}\epsilon_\theta(x_t, t)) + \sqrt{\beta_t}\epsilon$$

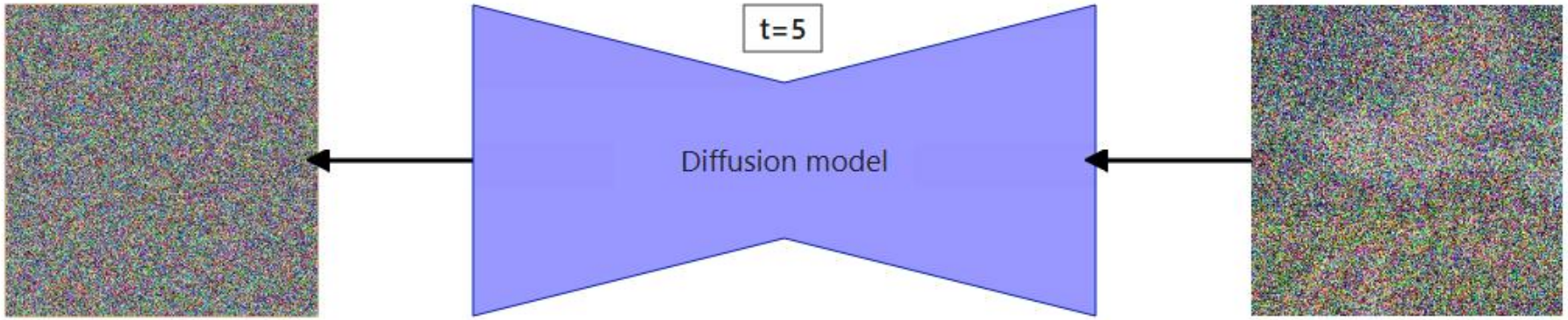
Reverse Diffusion Process





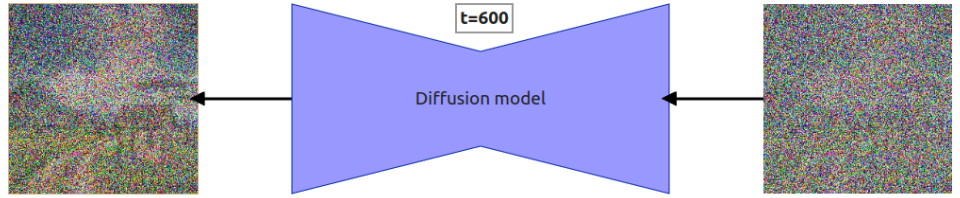
Reverse Diffusion Process

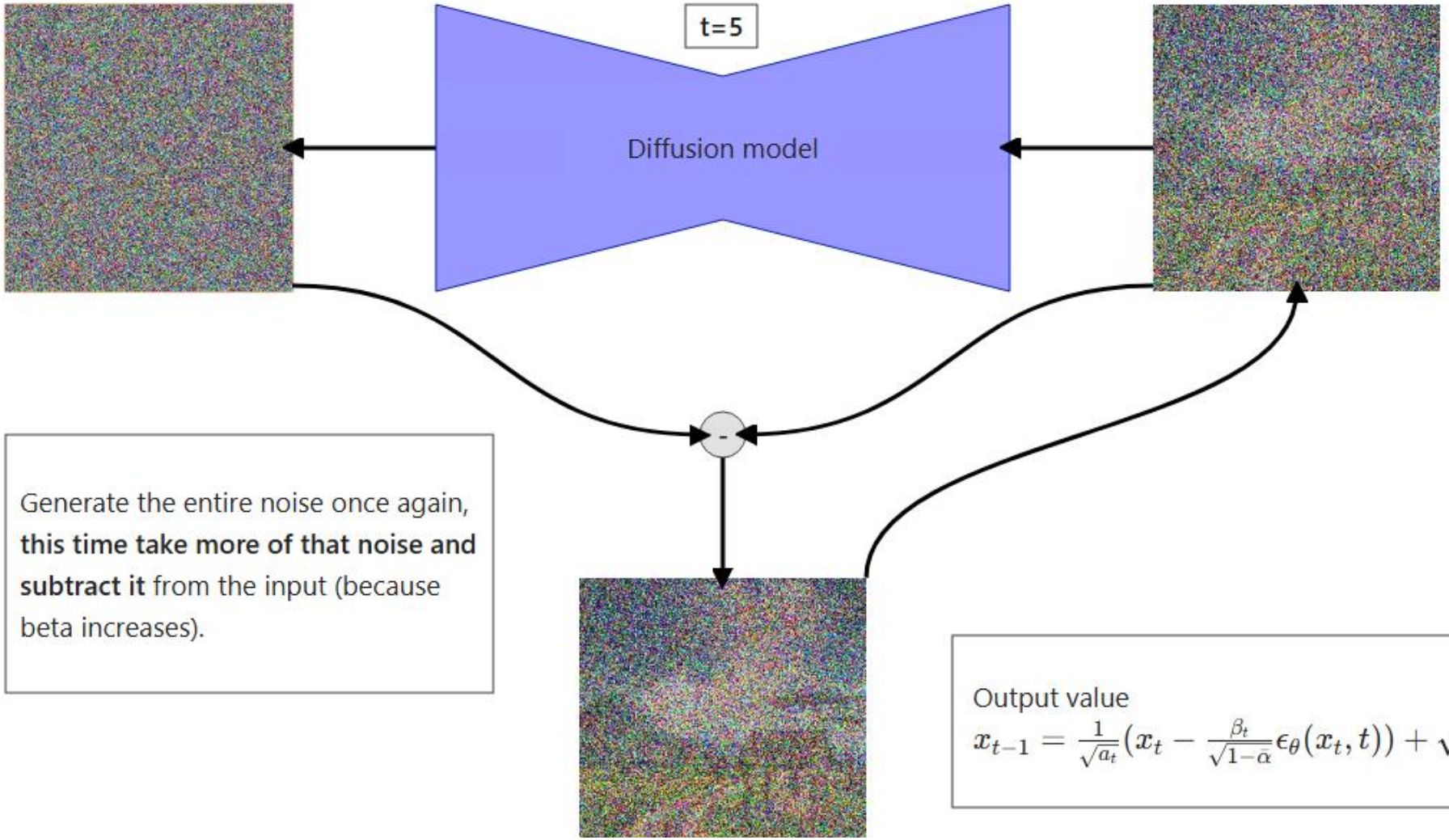




Generate the entire noise once again, **this time take more of that noise and subtract it** from the input (because beta increases).

Reverse Diffusion Process



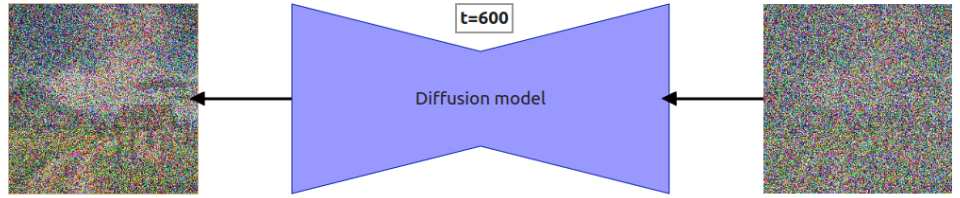


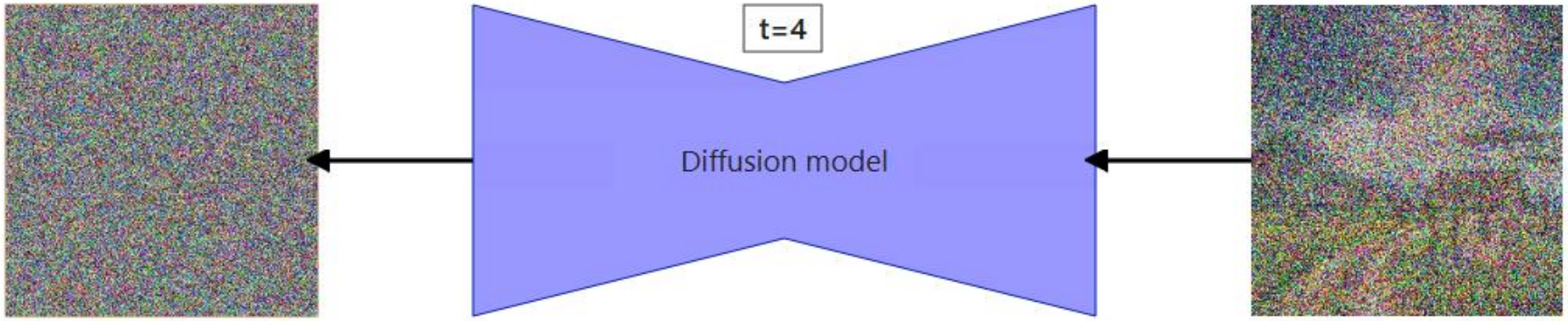
Generate the entire noise once again, **this time take more of that noise and subtract it** from the input (because beta increases).

Output value

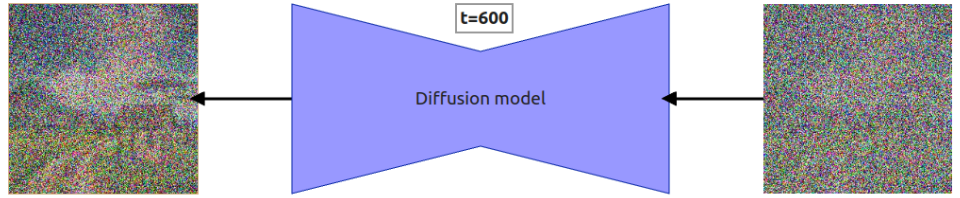
$$x_{t-1} = \frac{1}{\sqrt{a_t}}(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}}}\epsilon_\theta(x_t, t)) + \sqrt{\beta_t}\epsilon$$

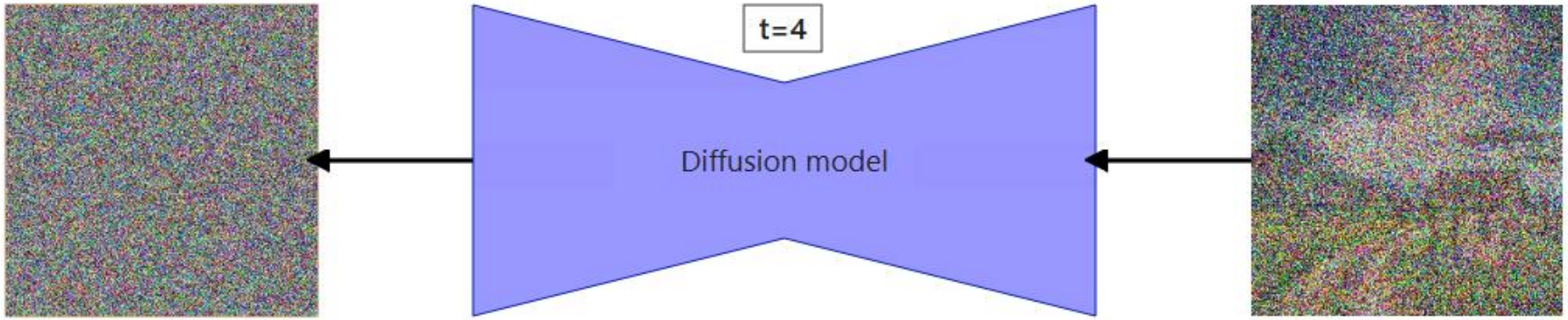
Reverse Diffusion Process





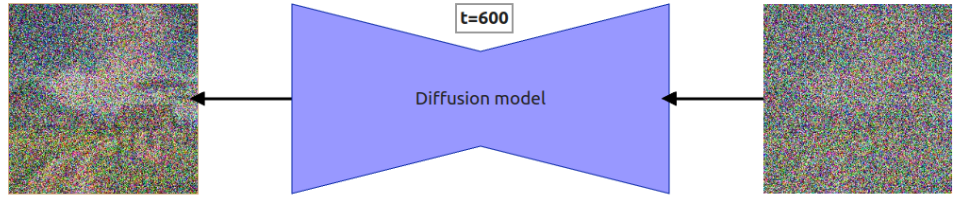
Reverse Diffusion Process

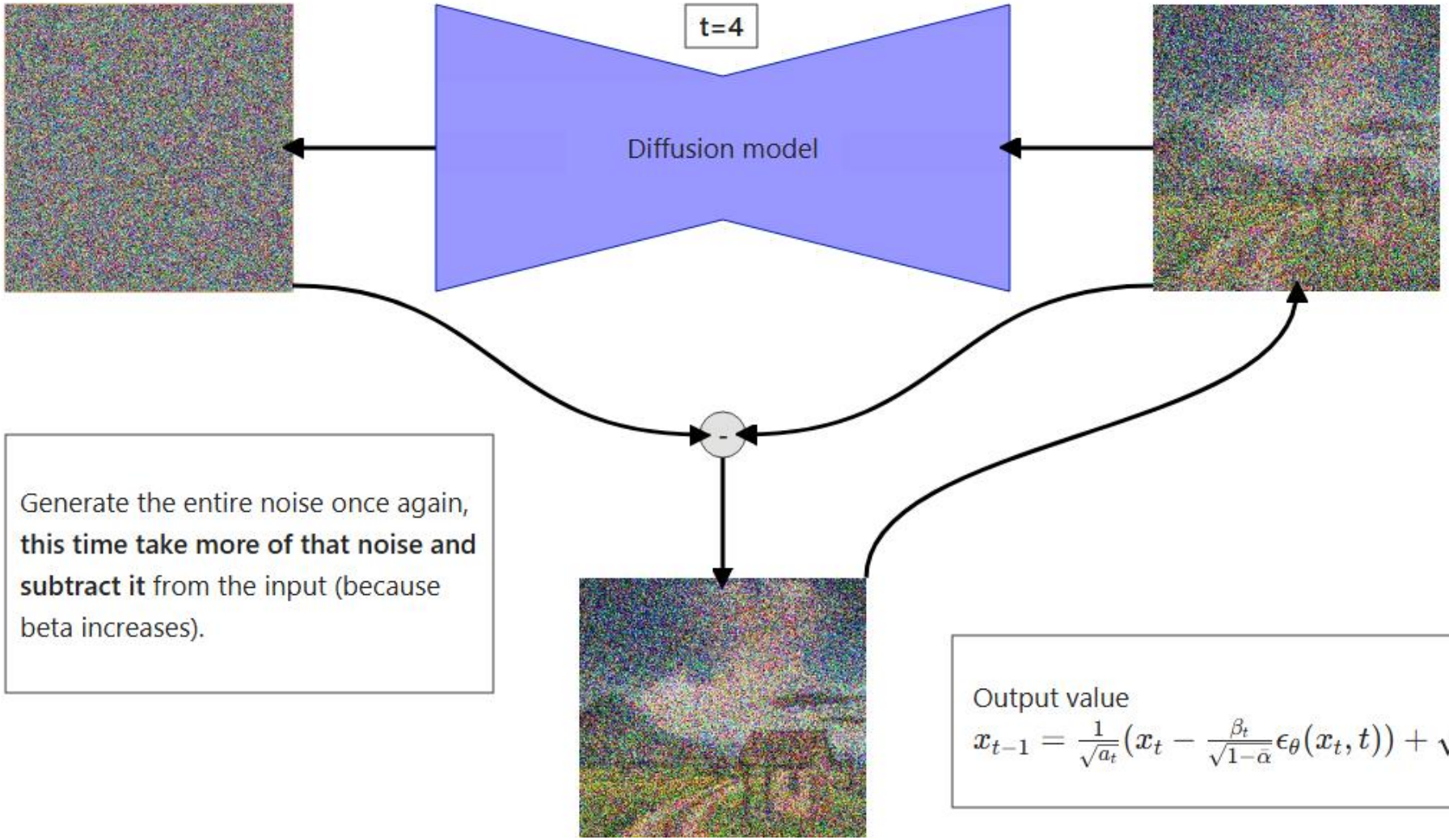




Generate the entire noise once again, **this time take more of that noise and subtract it** from the input (because beta increases).

Reverse Diffusion Process



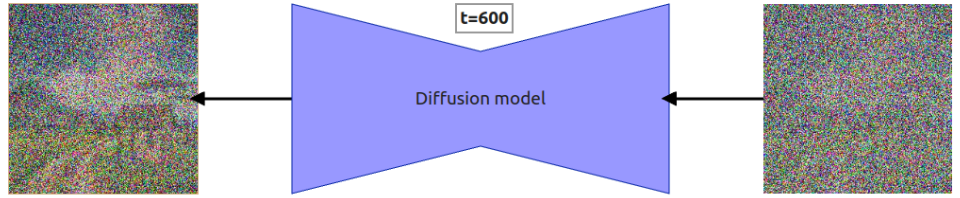


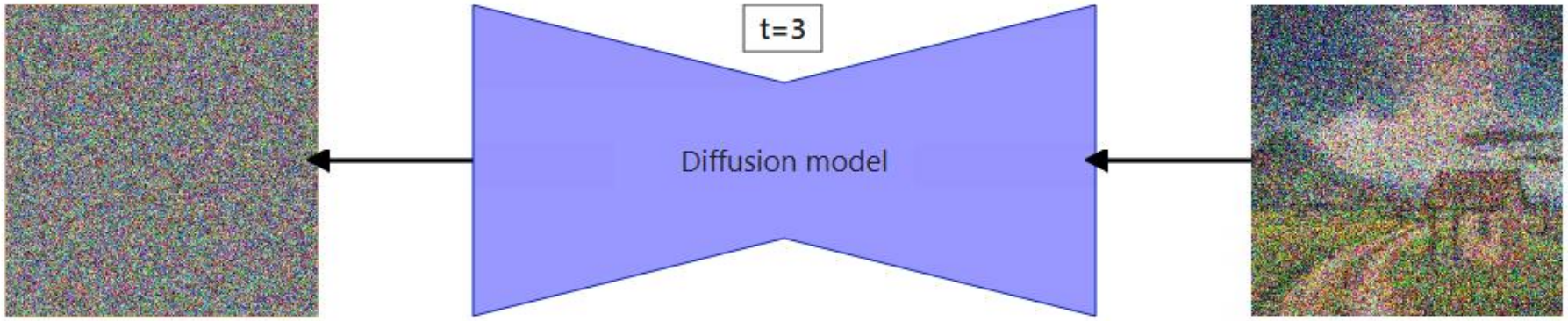
Generate the entire noise once again, **this time take more of that noise and subtract it** from the input (because beta increases).

Output value

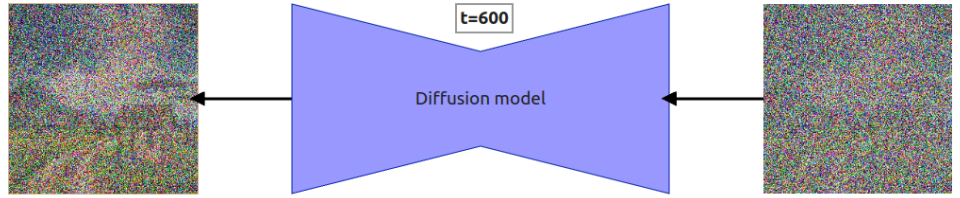
$$x_{t-1} = \frac{1}{\sqrt{a_t}}(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}}}\epsilon_\theta(x_t, t)) + \sqrt{\beta_t}\epsilon$$

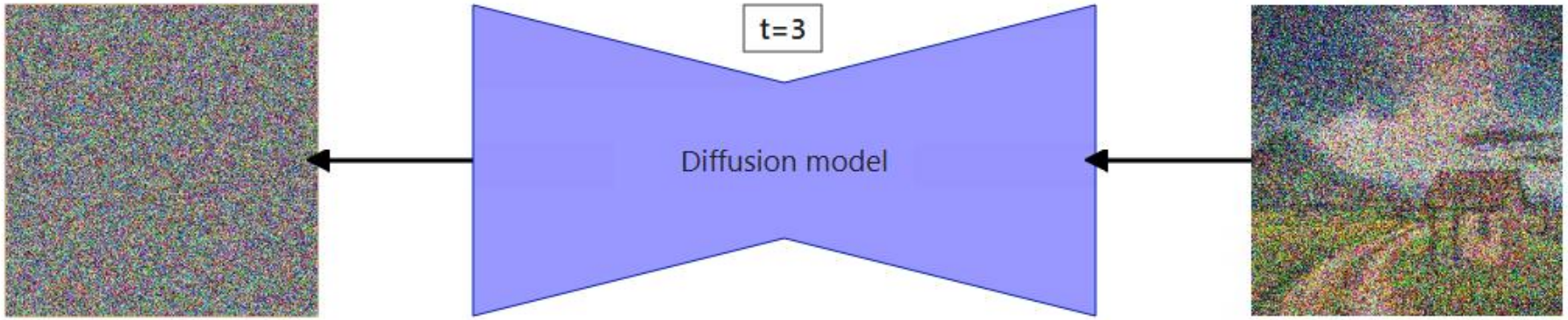
Reverse Diffusion Process





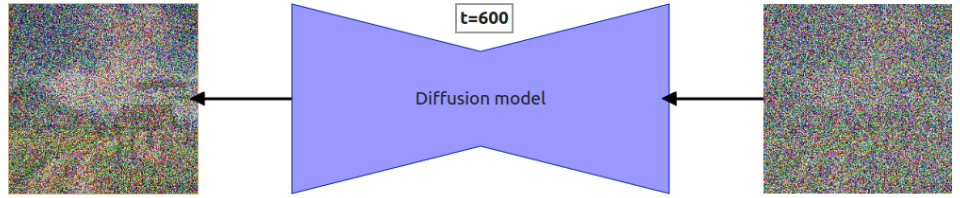
Reverse Diffusion Process

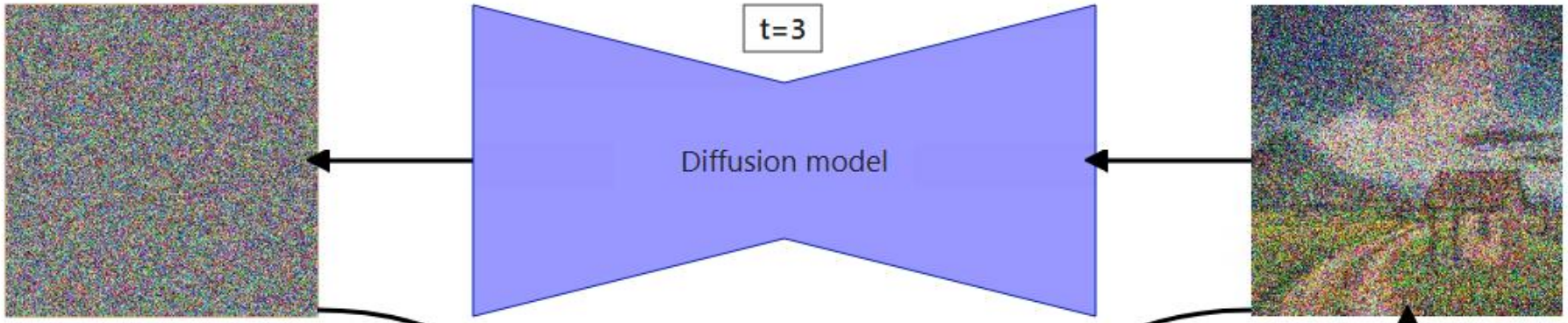




Generate the entire noise once again, **this time take more of that noise and subtract it** from the input (because beta increases).

Reverse Diffusion Process



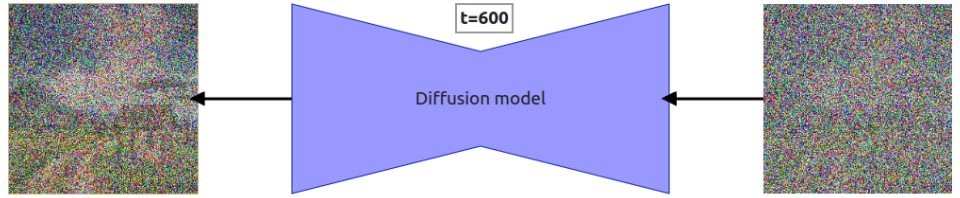


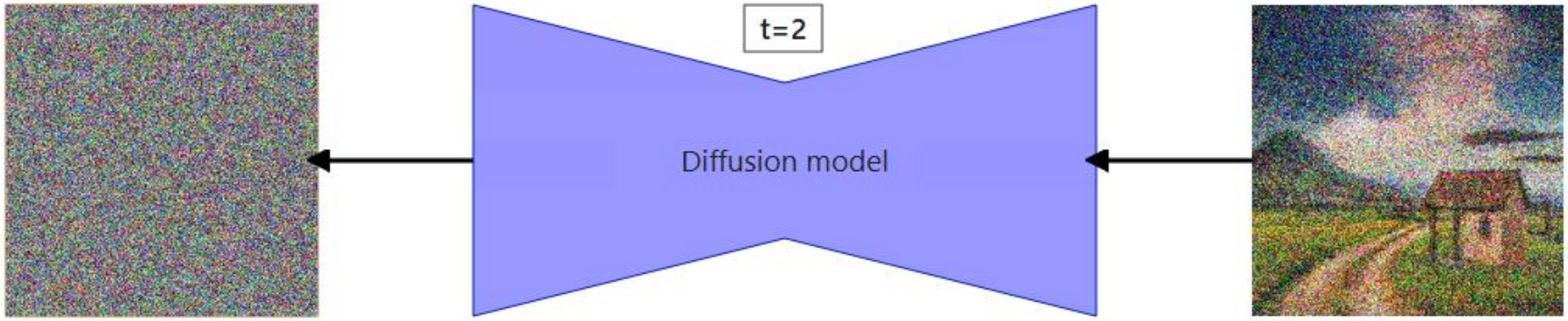
Generate the entire noise once again, **this time take more of that noise and subtract it** from the input (because beta increases).

Output value

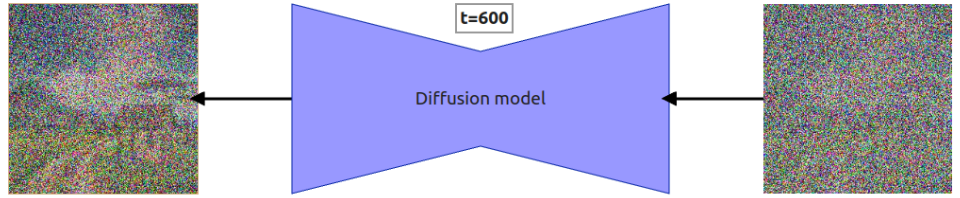
$$x_{t-1} = \frac{1}{\sqrt{a_t}}(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}}}\epsilon_\theta(x_t, t)) + \sqrt{\beta_t}\epsilon$$

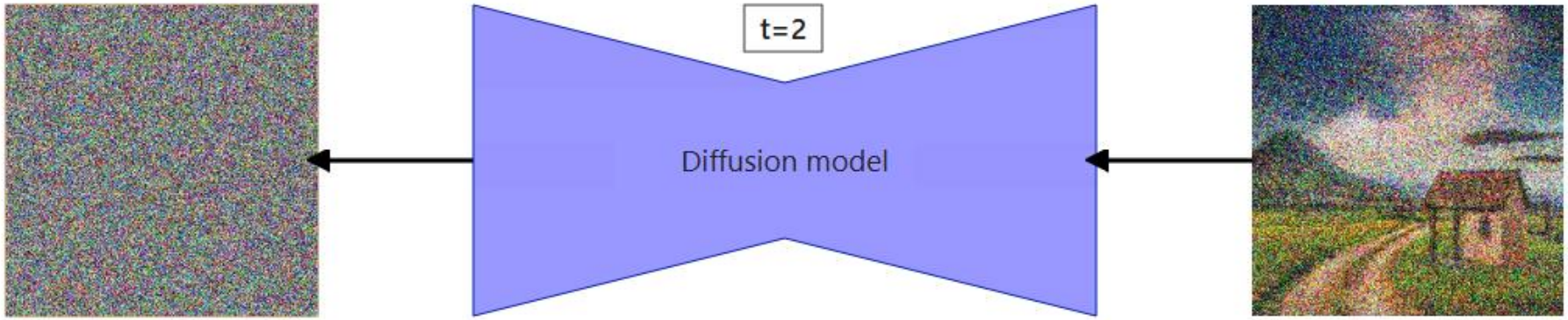
Reverse Diffusion Process





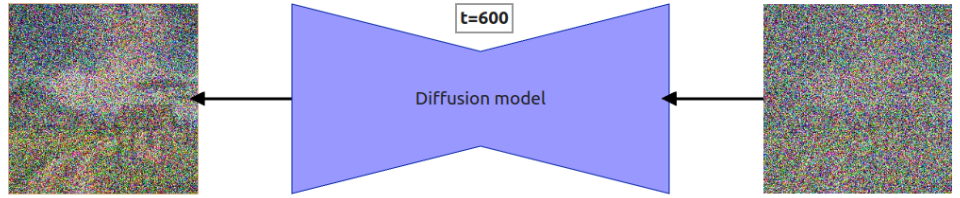
Reverse Diffusion Process

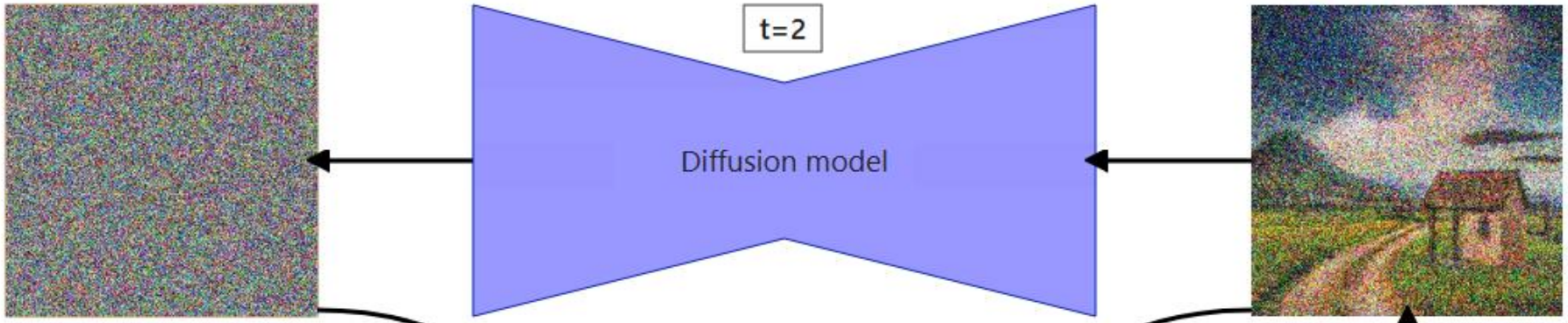




Generate the entire noise once again, **this time take more of that noise and subtract it** from the input (because beta increases).

Reverse Diffusion Process



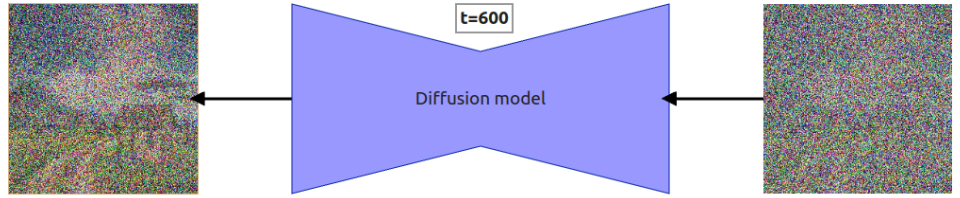


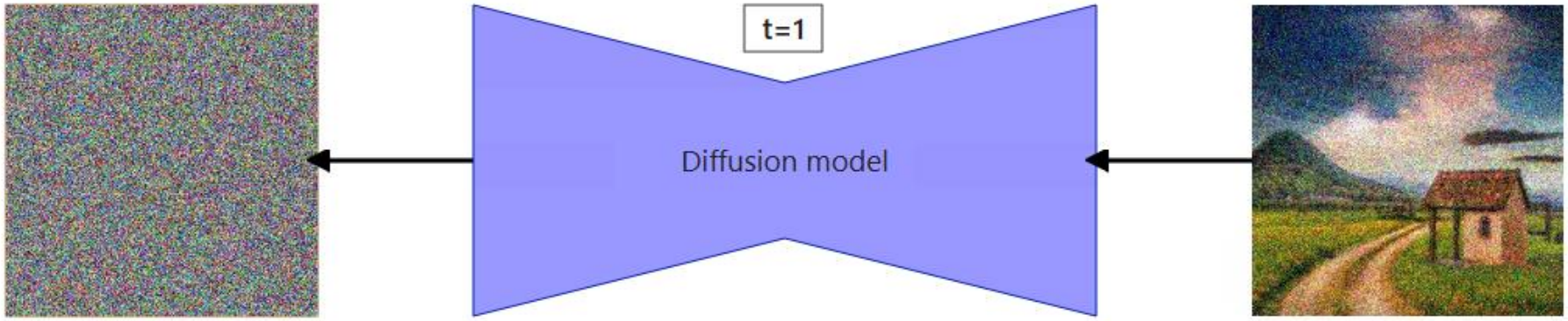
Generate the entire noise once again, **this time take more of that noise and subtract it** from the input (because beta increases).

Output value

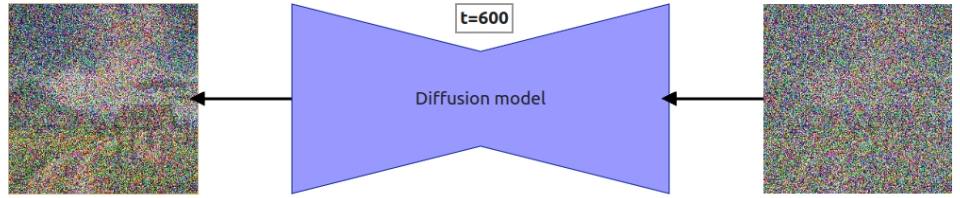
$$x_{t-1} = \frac{1}{\sqrt{a_t}}(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}}}\epsilon_\theta(x_t, t)) + \sqrt{\beta_t}\epsilon$$

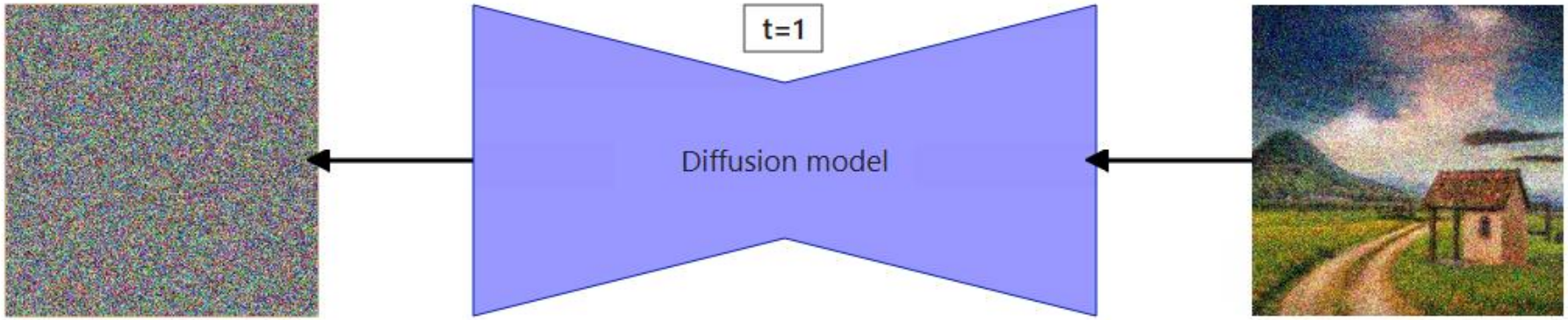
Reverse Diffusion Process





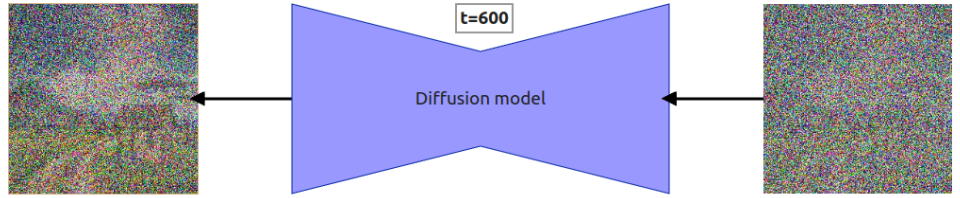
Reverse Diffusion Process

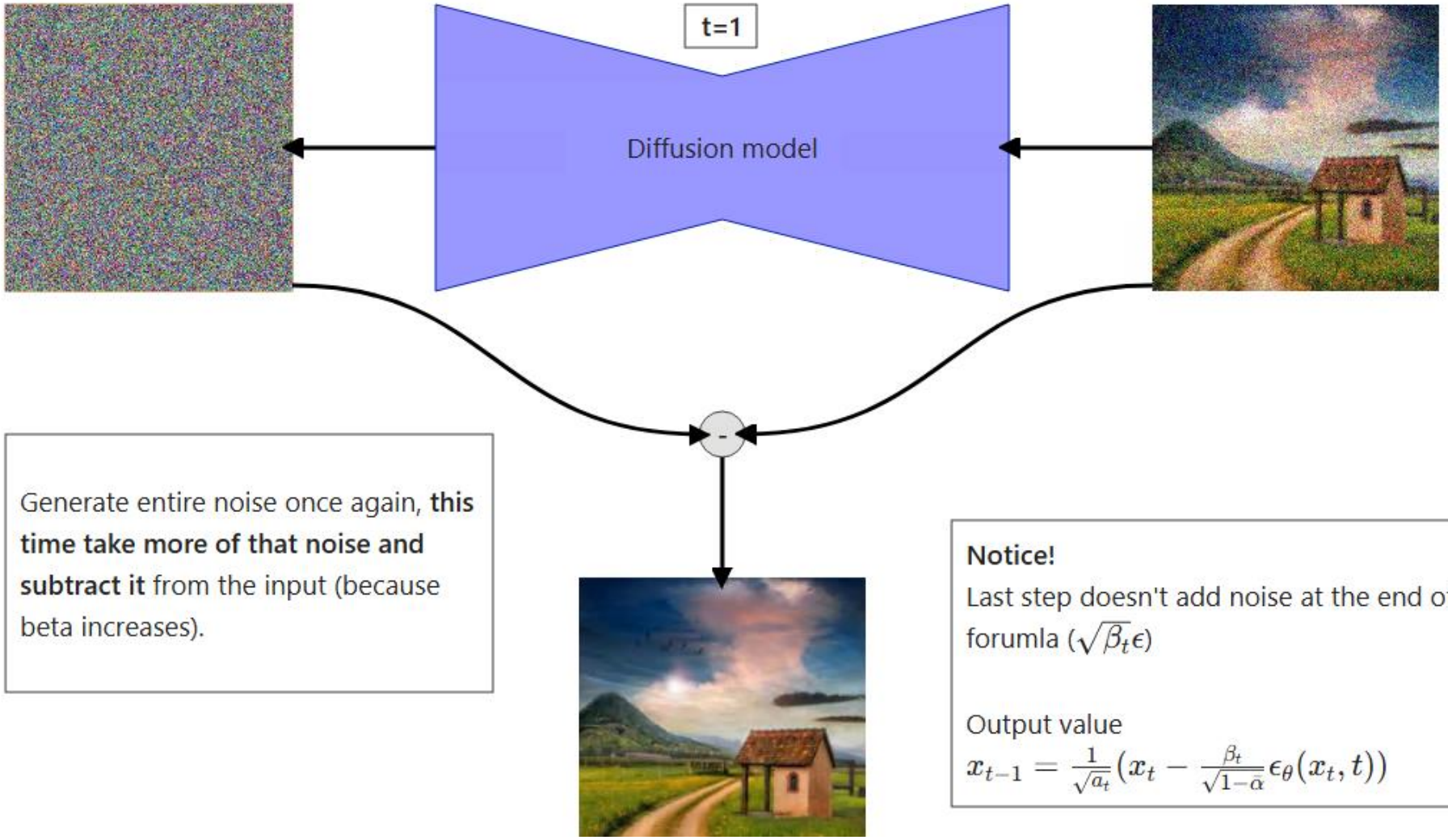




Generate entire noise once again, **this time take more of that noise and subtract it** from the input (because beta increases).

Reverse Diffusion Process



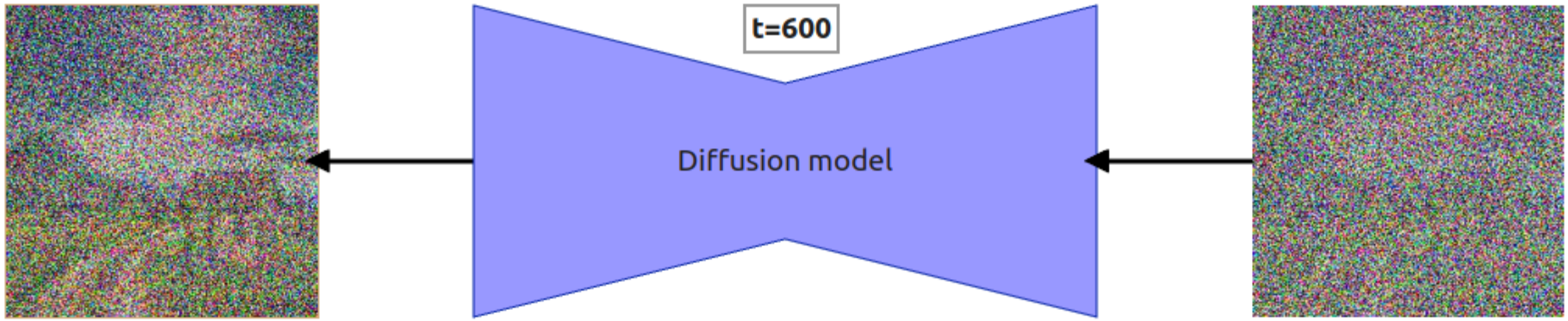


Generate entire noise once again, **this time take more of that noise and subtract it** from the input (because beta increases).

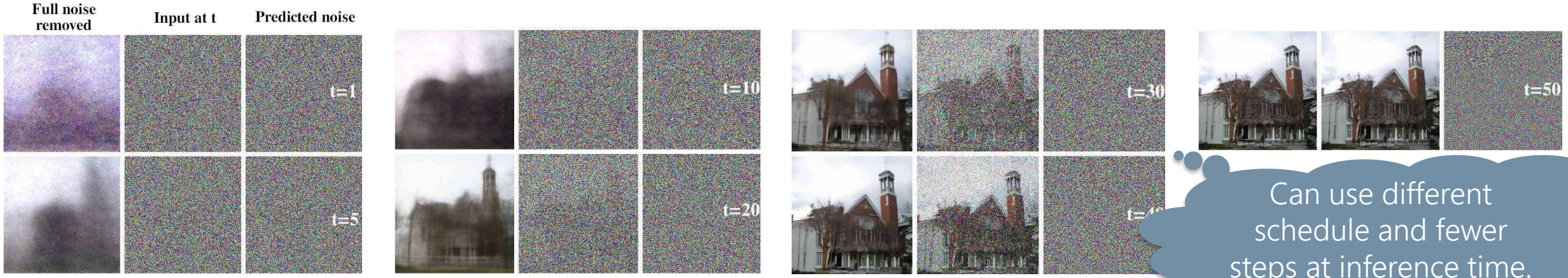
Notice!
Last step doesn't add noise at the end of the forumla ($\sqrt{\beta_t}\epsilon$)

Output value
$x_{t-1} = \frac{1}{\sqrt{a_t}}(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}}}\epsilon_\theta(x_t, t))$

Reverse Diffusion Process



The model has been trained to predict the <u>entire noise to be removed</u> at a given timestep, e.g., 600 in this picture, to recover the original image.



Can use different schedule and fewer steps at inference time.

POLITECNICO MILANO 1863

Lil'Log | What are Diffusion Models?

Date: July 11, 2021 | Estimated Reading Time: 32 min | Author: Lilian Weng

https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

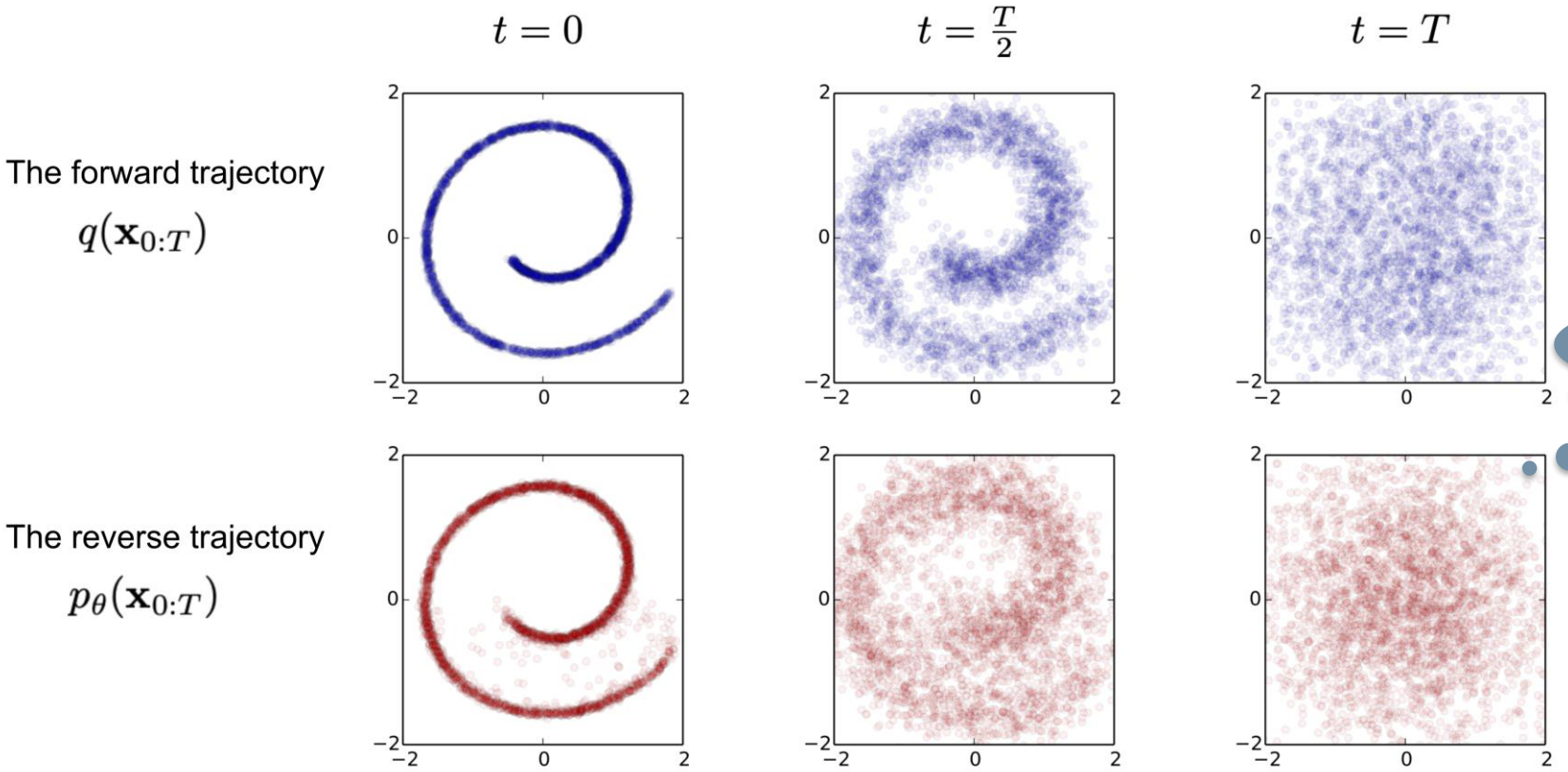
Conditioned Generation

Matteo Matteucci, PhD (matteo.matteucci@polimi.it)
Artificial Intelligence and Robotics Laboratory
Politecnico di Milano

AIRLAB
ARTIFICIAL INTELLIGENCE AND ROBOTICS LAB

One "class" one model …

An example of training a diffusion model for modeling a 2D swiss roll data. (image adapted from Sohl-Dickstein et al., 2015)



Different seeds generate different swiss rolls … but just rolls.

Classifier Guided Diffusion

To condition on different classes, Dhariwal & Nichol (2021) trained a classifier $f_\phi(y|x_t, t)$ and used gradients $\nabla_x \log f_\phi(y|x_t)$ to guide the diffusion sampling process toward conditioning information y.

Score function of the joint distribution …

$$\begin{aligned}
\nabla_{\mathbf{x}_t} \log q(\mathbf{x}_t, y) &= \nabla_{\mathbf{x}_t} \log q(\mathbf{x}_t) + \nabla_{\mathbf{x}_t} \log q(y|\mathbf{x}_t) \\
&\approx -\frac{1}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) + \nabla_{\mathbf{x}_t} \log f_\phi(y|\mathbf{x}_t) \\
&= -\frac{1}{\sqrt{1-\bar{\alpha}_t}} (\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) - \sqrt{1-\bar{\alpha}_t} \nabla_{\mathbf{x}_t} \log f_\phi(y|\mathbf{x}_t))
\end{aligned}$$

The classified guided noise predictor becomes

$$\bar{\epsilon}_\theta(x_t, t) = \epsilon_\theta(x_t, t) - w\sqrt{1-\bar{\alpha}_t} \nabla_{x_t} \log f_\phi(y|x_t)$$

Classifier Guided Diffusion

The resulting Ablated Diffusion Model (ADM) and the one with additional classifier guidance (ADM-G) are able to achieve better results than SOTA generative models (e.g. BigGAN).

Algorithm 2 Classifier guided DDIM sampling, given a diffusion model $\epsilon_\theta(x_t)$, classifier $f_\phi(y|x_t)$, and gradient scale s.

Input: class label y, gradient scale s
$x_T \leftarrow$ sample from $\mathcal{N}(0, \mathbf{I})$
for all t from T to 1 **do**
  $\hat{\epsilon} \leftarrow \epsilon_\theta(x_t) - \sqrt{1-\bar{\alpha}_t}\,\nabla_{x_t} \log f_\phi(y|x_t)$
  $x_{t-1} \leftarrow \sqrt{\bar{\alpha}_{t-1}}\left(\frac{x_t - \sqrt{1-\bar{\alpha}_t}\hat{\epsilon}}{\sqrt{\bar{\alpha}_t}}\right) + \sqrt{1-\bar{\alpha}_{t-1}}\hat{\epsilon}$
end for
return x_0

The diffusion model and the classifier are given at sampling time

Note: DDIM stands for Denoising Diffusion Implicit Model

Classifier Free Guidance (Training)

It is possible to run conditional diffusion steps using the scores from a conditional and an unconditional diffusion model (Ho & Salimans, 2021).

Let $p_\theta(x|\emptyset)$ be parameterized through a score estimator $\epsilon_\theta(x_t, t)$ and $p_\theta(x|y)$ through $\epsilon_\theta(x_t, t, y)$ learned jointly via a single neural network.

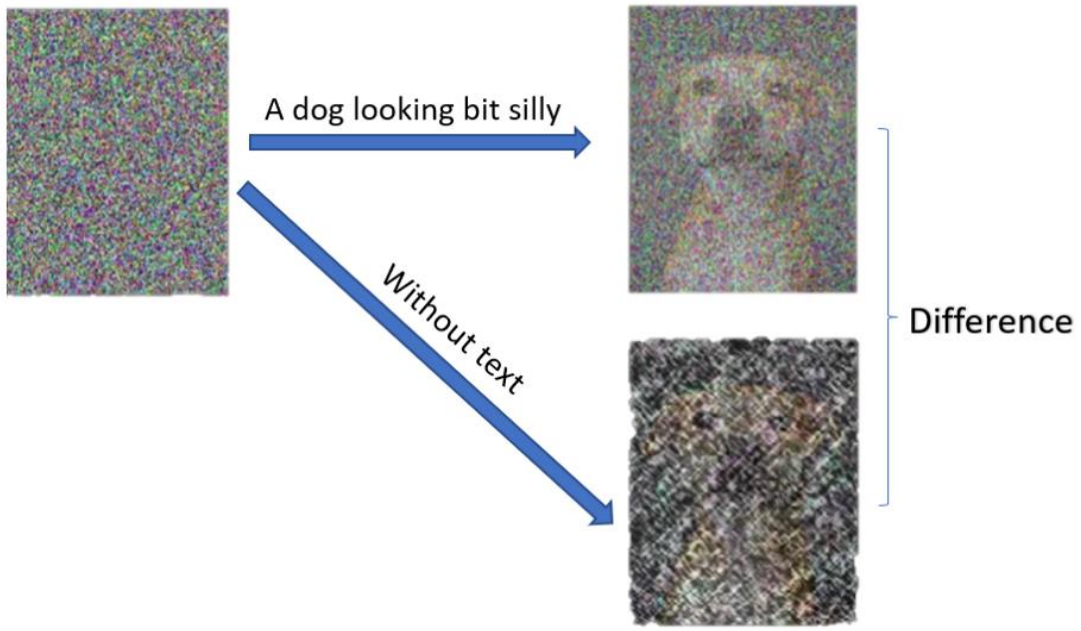
During training, randomly replace y with a learnable null token $\emptyset$ at random such that the model knows how to generate images unconditionally as well, i.e., $\epsilon_\theta(x_t, t) = \epsilon_\theta(x_t, t, y = \emptyset)$.

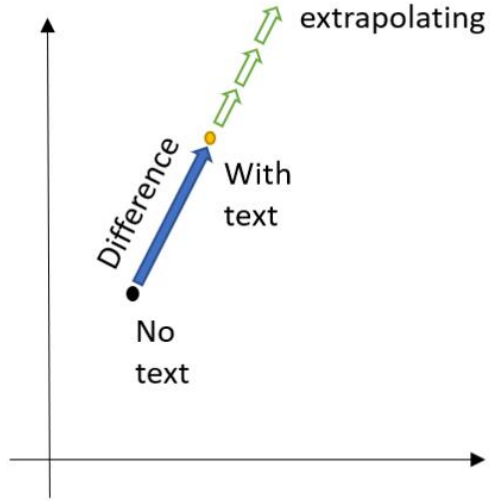
Classifier Free Guidance (Sampling)

$$
\begin{aligned}
\nabla_{\mathbf{x}_t} \log p(y|\mathbf{x}_t) &= \nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t|y) - \nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t) \\
&= -\frac{1}{\sqrt{1-\bar{\alpha}_t}} \Big(\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, y) - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \Big) \\
\bar{\boldsymbol{\epsilon}}_\theta(\mathbf{x}_t, t, y) &= \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, y) - \sqrt{1-\bar{\alpha}_t}\, w \nabla_{\mathbf{x}_t} \log p(y|\mathbf{x}_t) \\
&= \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, y) + w\big(\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, y) - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\big) \\
&= (w+1)\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, y) - w\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)
\end{aligned}
$$

Make one prediction with class token, one prediction with null token, and combine the predictions …

Classifier Free Guidance (Sampling)





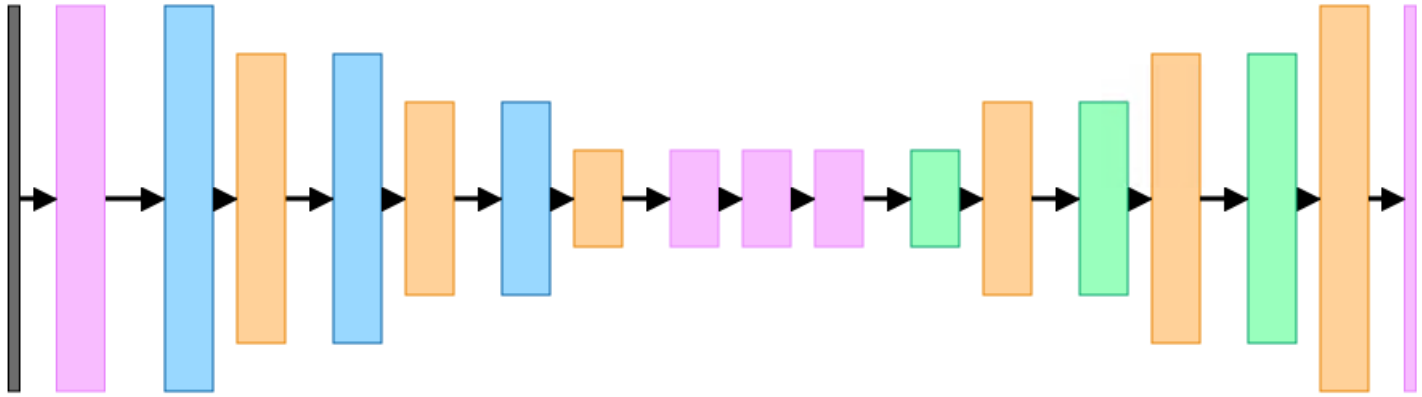
Make one prediction with class token, one prediction with null token, and combine the predictions …

Kemal Erdem, (Nov 2023). "Step by Step visual introduction to Diffusion Models.".
https://erdem.pl/2023/11/step-by-step-visual-introduction-to-diffusion-models

Model Architecture

Matteo Matteucci, PhD (matteo.matteucci@polimi.it)
Artificial Intelligence and Robotics Laboratory
Politecnico di Milano

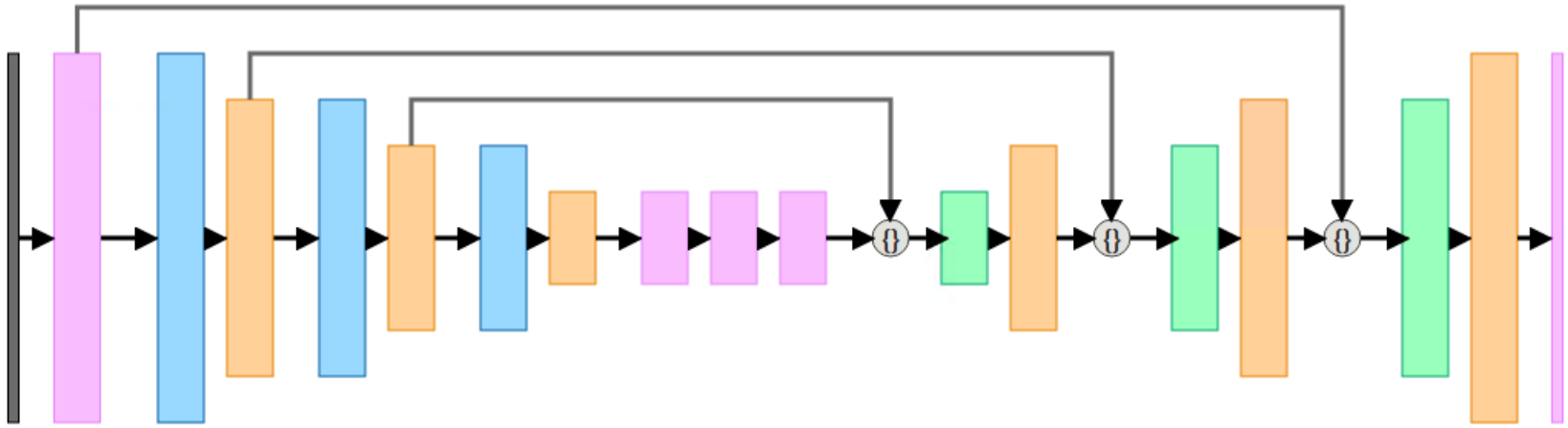
Diffusion Model Architecture (U-Net backbone)



Input
Conv Block
Downsample Block
Self-Attention Block
Upsample Block
Embedding vector

We start with the modified **U-Net architecture**.
Basic ResNet Blocks were replaced by either Self-Attention blocks or modified ResNet blocks.

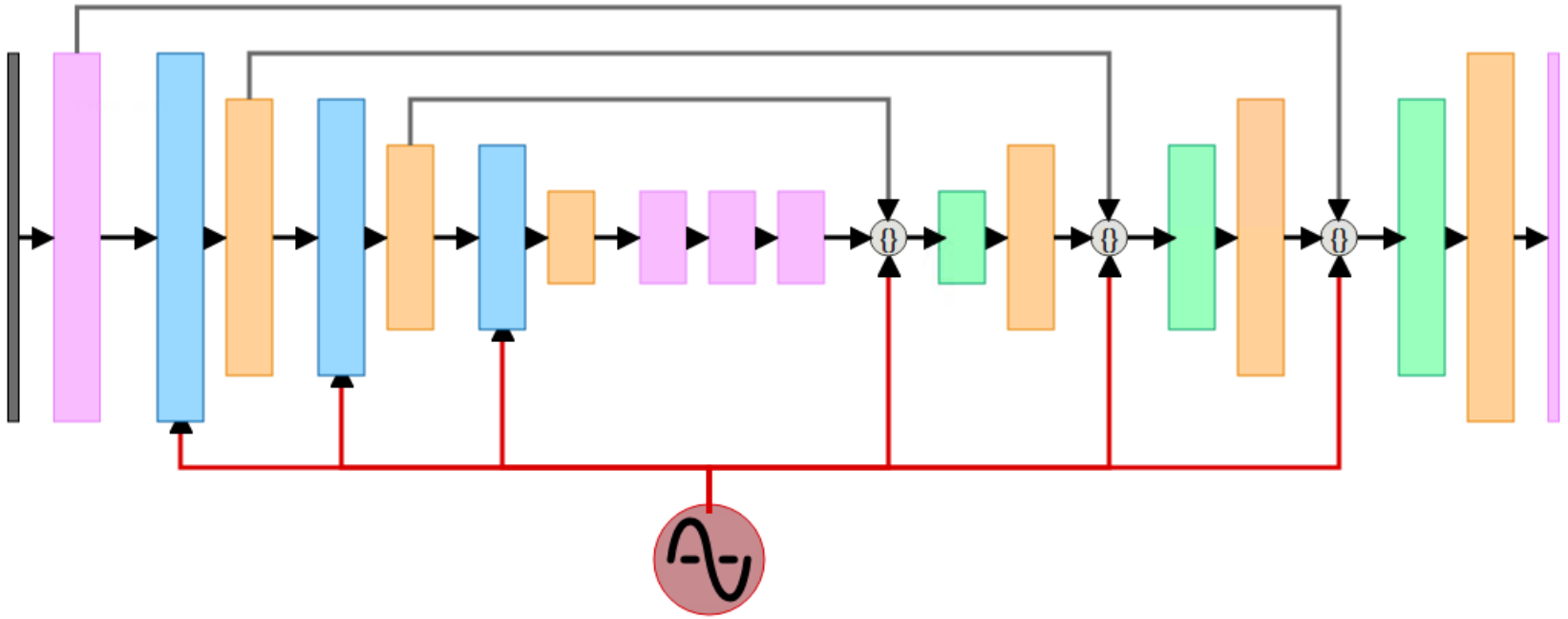
Diffusion Model Architecture (U-Net backbone)



Input
Conv Block
Downsample Block
Self-Attention Block
Upsample Block
Embedding vector

To prevent information lose, we're adding **skip connections to the upsampling blocks**. Notice where those connections are coming from.

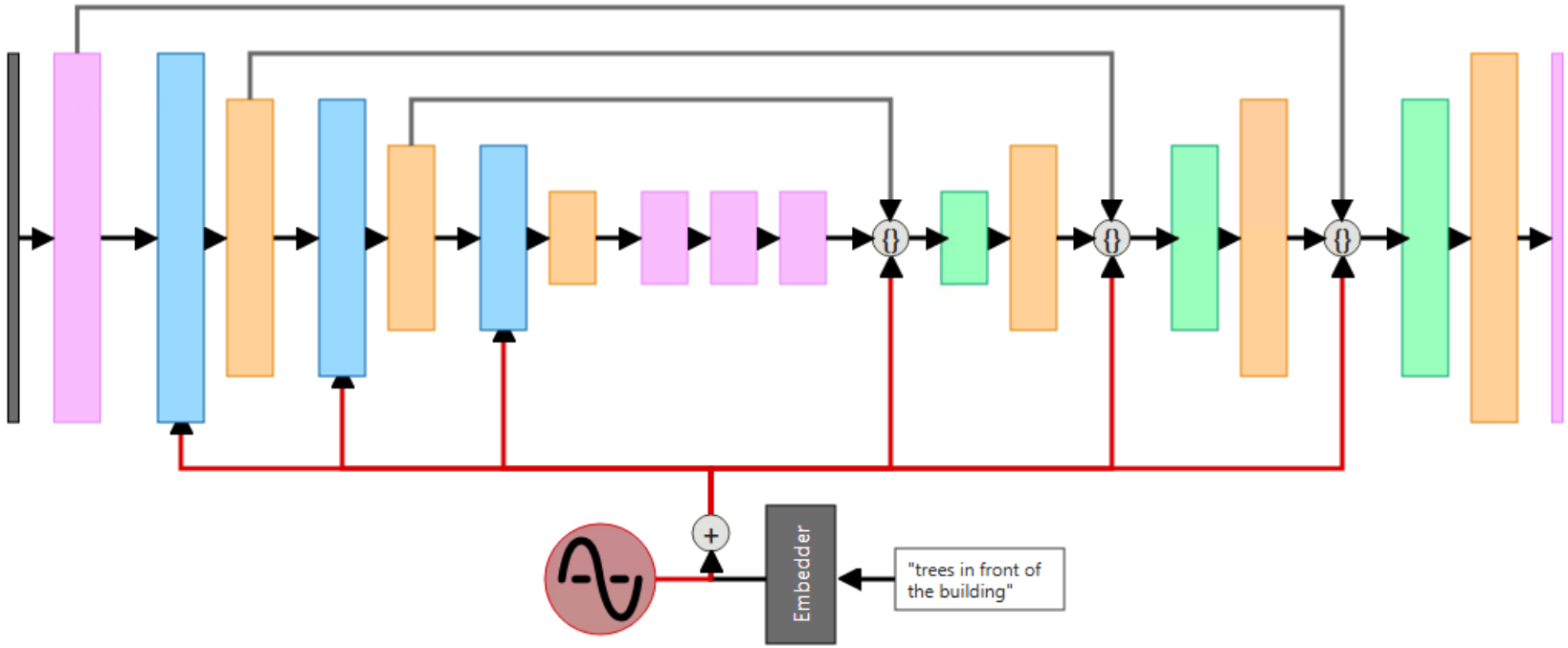
Diffusion Model Architecture (U-Net backbone)



Input
Conv Block
Downsample Block
Self-Attention Block
Upsample Block
Embedding vector

The next step is to add information about the current timestep **t**. To do that we're using **sinusoidal embedding** and that information is added to **all downsample and upsample** blocks.

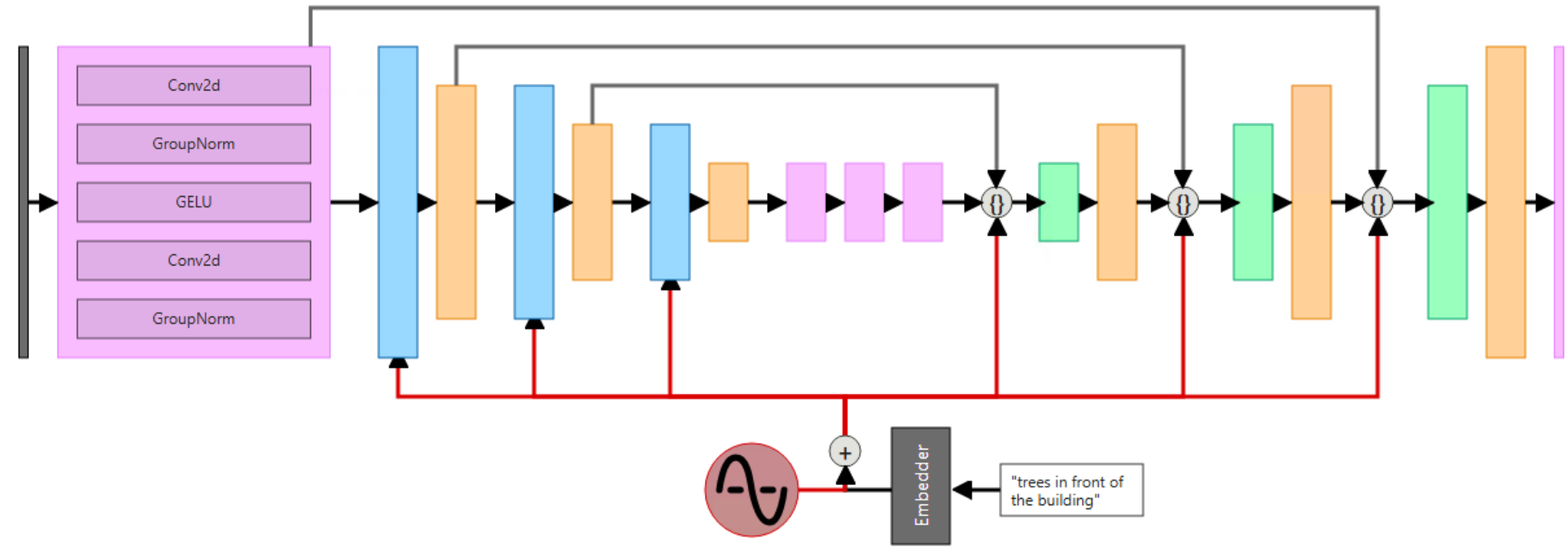
Diffusion Model Architecture (U-Net backbone)



Input
Conv Block
Downsample Block
Self-Attention Block
Upsample Block
Embedding vector

To make the network conditional (dependent on the external input), we're adding a **text embedder**. It will create embedding for given text and **add output vector to the timestep embedding**.

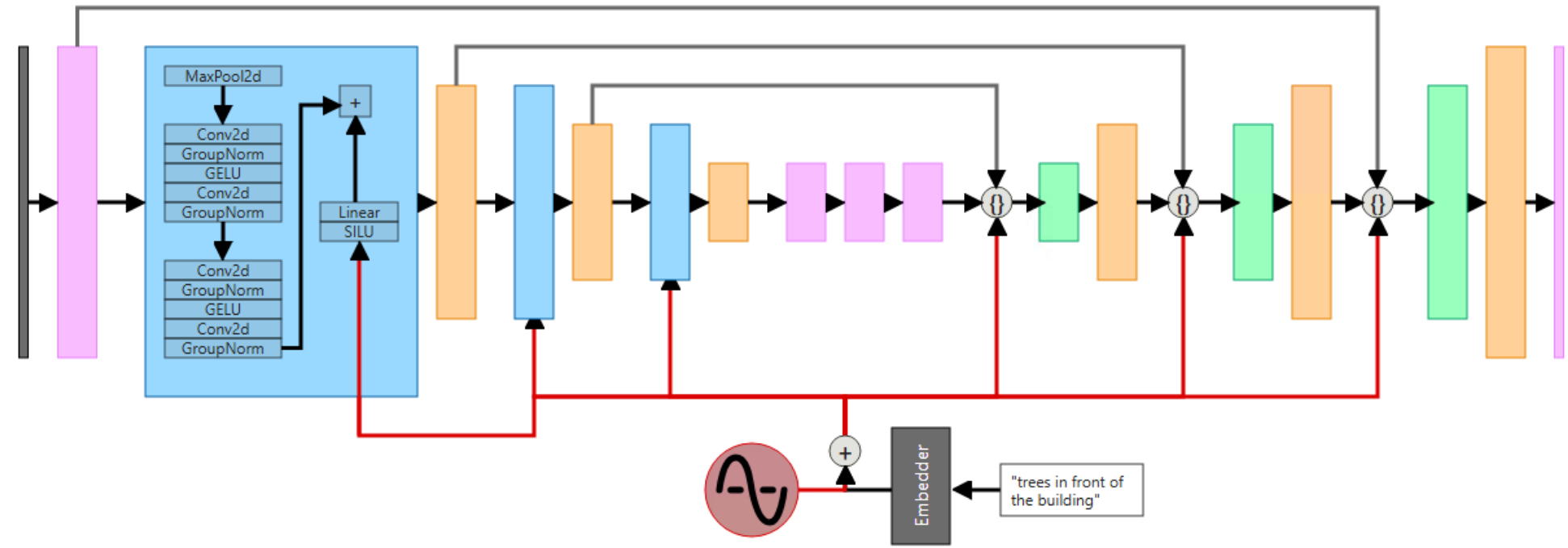
Diffusion Model Architecture (U-Net backbone)



Input
Conv Block
Downsample Block
Self-Attention Block
Upsample Block
Embedding vector

Detailed exploded view of the ResNet/Conv Block. All blocks are structured in the same way, only tensor shapes are different.

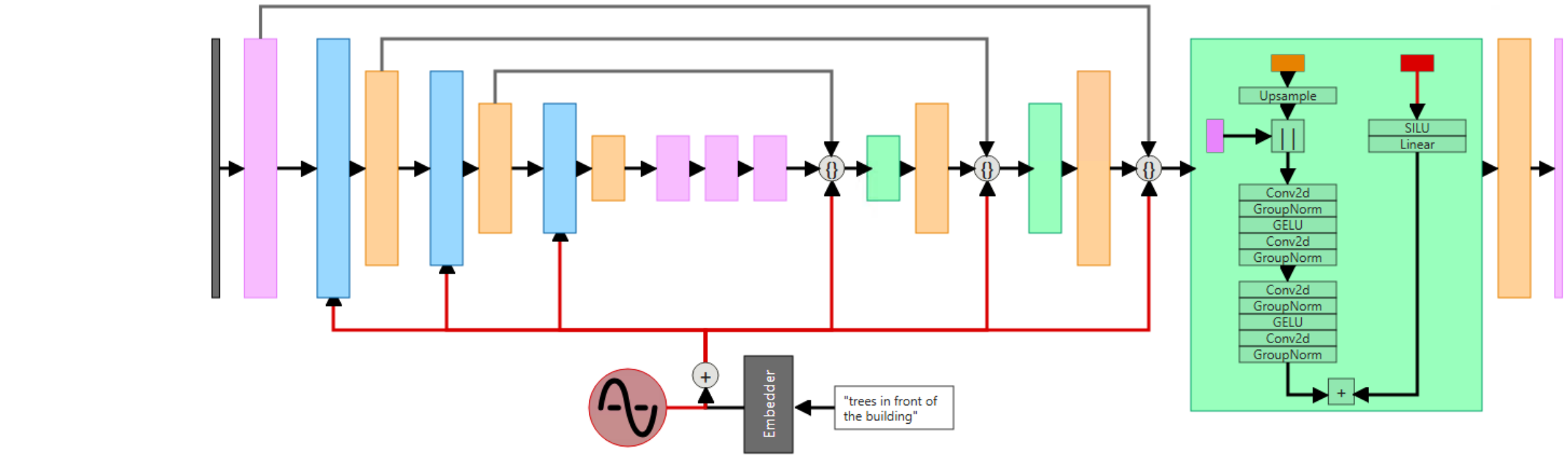
Diffusion Model Architecture (U-Net backbone)



Input
Conv Block
Downsample Block
Self-Attention Block
Upsample Block
Embedding vector

Detailed exploded view of the Downsample Block. All blocks are structured in the same way, only tensor shapes are different.

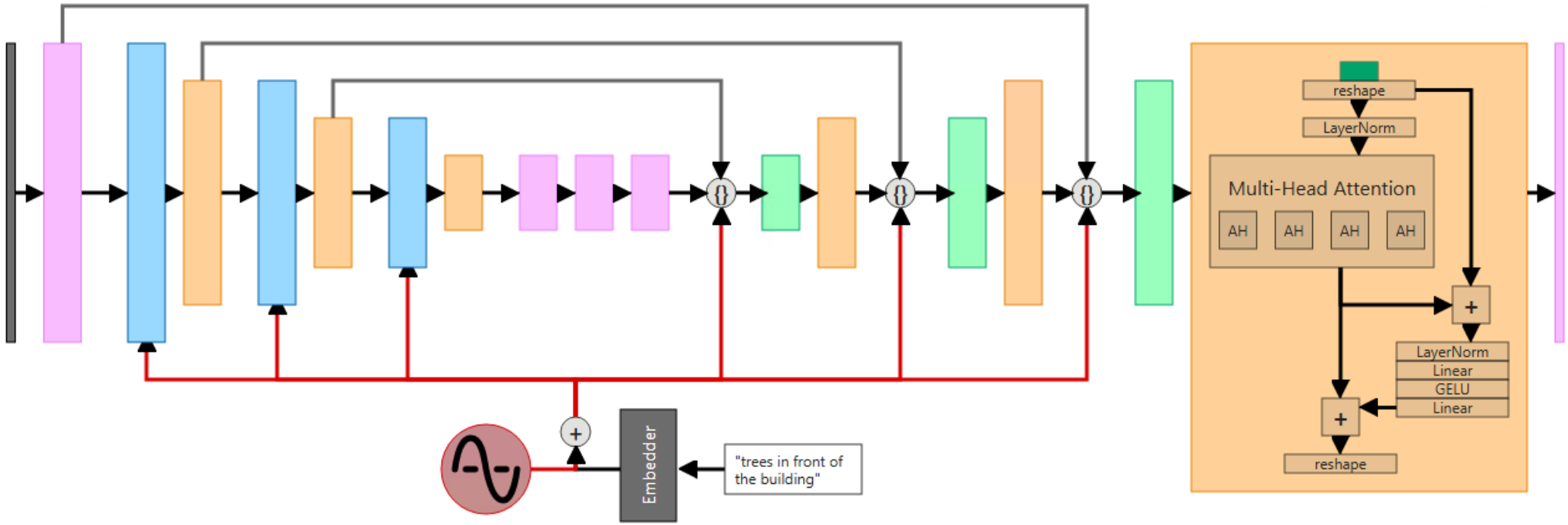
Diffusion Model Architecture (U-Net backbone)



Input
Conv Block
Downsample Block
Self-Attention Block
Upsample Block
Embedding vector

Detailed exploded view of the Upsample Block. All blocks are structured in the same way, only tensor shapes are different.

Diffusion Model Architecture (U-Net backbone)



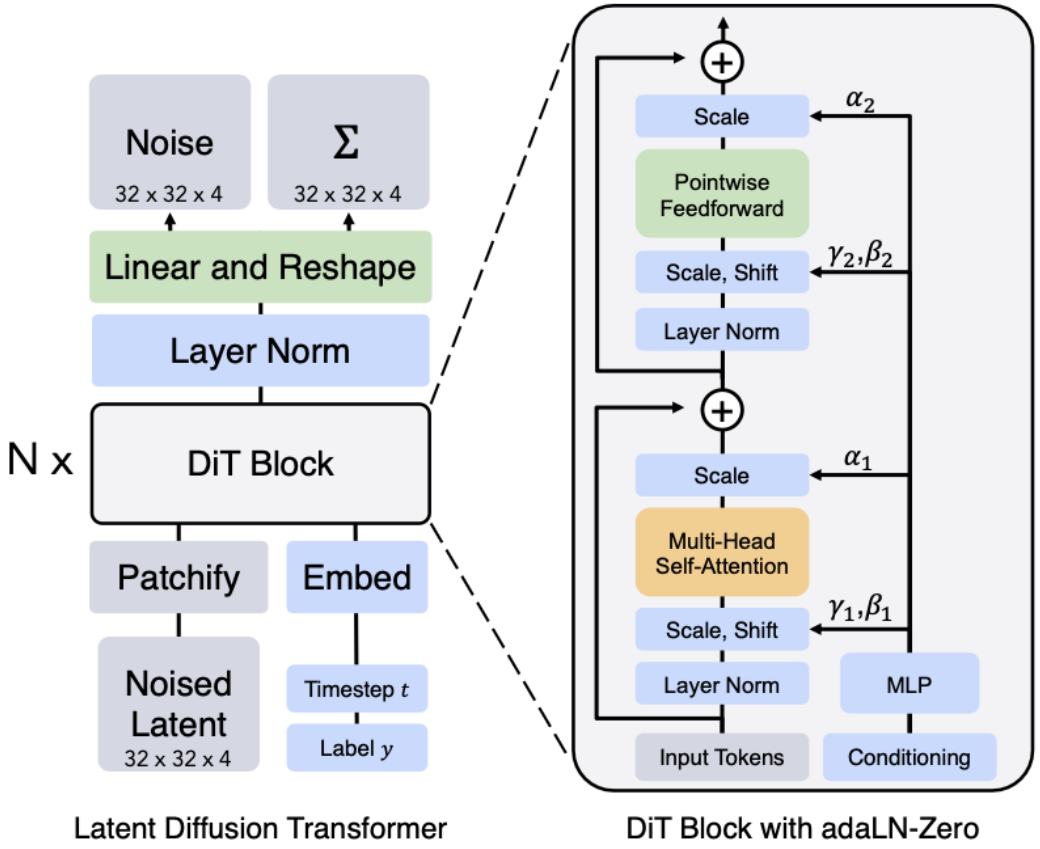
Input
Conv Block
Downsample Block
Self-Attention Block
Upsample Block
Embedding vector

Detailed exploded view of the Attention Block. All blocks are structured in the same way, only tensor shapes are different.

Diffusion Model Architecture (Transformer Backbone)

The **Diffusion Transformer (DiT)** operates on latent patches using the same design space of the Latent Diffusion Model (Peebles & Xie, 2023)

- Take latent representation z of input
- "Patchify" noise latent of size $I \times I \times C$ into patches of size p converting it into a sequence of $(I/p)^2$ patches
- This sequence of tokens goes through Transformer blocks using adaLN for conditioning on time and label
- The transformer decoder outputs noise predictions and an output diagonal covariance prediction.



POLITECNICO MILANO 1863

Lil'Log | What are Diffusion Models?

Date: July 11, 2021 | Estimated Reading Time: 32 min | Author: Lilian Weng

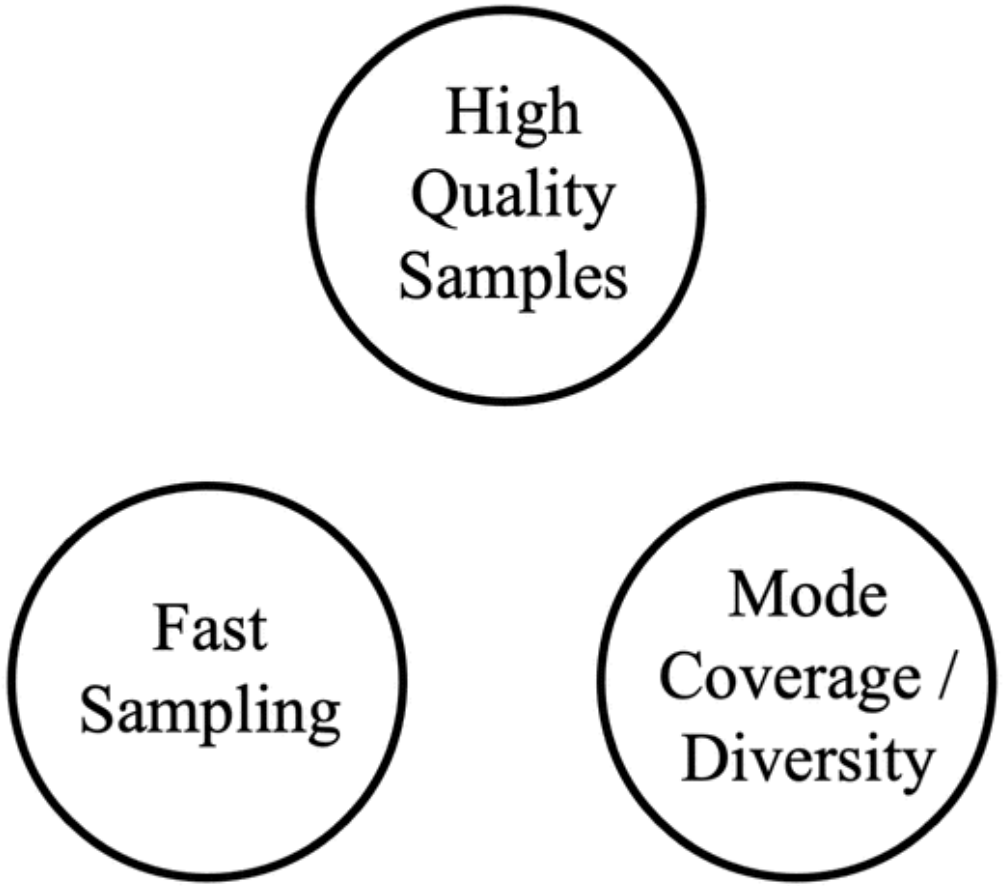
https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

Improving DDPM Sampling

Matteo Matteucci, PhD (matteo.matteucci@polimi.it)
Artificial Intelligence and Robotics Laboratory
Politecnico di Milano

AIRLAB
ARTIFICIAL INTELLIGENCE AND ROBOTICS LAB

The Generative Trilemma



Speeding up Diffusion Models

"For example, it takes around 20 hours to sample 50k images of size 32 × 32 from a DDPM, but less than a minute to do so from a GAN on an Nvidia 2080 Ti GPU." Song et al. (2020)

One way is to run a "strided" sampling schedule (Nichol & Dhariwal, 2021) by taking the sampling update every $\lceil T/S \rceil$ steps. The new sampling schedule become $\{\tau_1, \dots, \tau_S\}$ with $\tau_1 < \tau_2 < \cdots < \tau_S \in [1, T]$ and $S < T$.

A more effective approach is the one proposed by Song et al. (2020) which is named *Denoising Diffusion Implicit Model (DDIM)* …

Denoising Diffusion Implicit Model

Let's rewrite $q_\sigma(x_t - 1|x_t, x_0)$ to be parameterized by a desired standard deviation σ_t. It can be proved that:

$$x_{t-1} = \sqrt{\bar{\alpha}_{t-1}}x_0 + \sqrt{1-\bar{\alpha}_{t-1}}\,\epsilon_{t-1}$$
$$= \sqrt{\bar{\alpha}_{t-1}}x_0 + \sqrt{1-\bar{\alpha}_{t-1}-\sigma^2}\epsilon_t + \sigma_t\epsilon$$
$$= \sqrt{\bar{\alpha}_{t-1}}\left(\frac{x_t - \sqrt{1-\bar{\alpha}_{t-1}}\,\epsilon_\theta^{(t)}(x_t)}{\sqrt{\bar{\alpha}_t}}\right) + \sqrt{1-\bar{\alpha}_{t-1}-\sigma^2}\epsilon_\theta^{(t)}(x_t) + \sigma_t\epsilon$$

from this we get

$$q(x_{t-1}|x_t, x_0) = N\left(x_{t-1}; \sqrt{\bar{\alpha}_{t-1}}\left(\frac{x_t - \sqrt{1-\bar{\alpha}_{t-1}}\,\epsilon_\theta^{(t)}(x_t)}{\sqrt{\bar{\alpha}_t}}\right) + \sqrt{1-\bar{\alpha}_{t-1}-\sigma^2}\epsilon_\theta^{(t)}(x_t), \sigma^2 I\right)$$

where the model $\epsilon_\theta^{(t)}$ predicts the ϵ_t from x_t.

Denoising Diffusion Implicit Model

Recall that in $q(x_{t-1}|x_t, x_0) = N\left(x_{t-1}; \mu(x_t, x_0), \tilde{\beta} I\right)$ therefore we have

$$\tilde{\beta}_t = \sigma_t^2 = \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t} \cdot \beta_t$$

Let $\sigma_t^2 = \eta \tilde{\beta}_t$ where $\eta = 0$ makes sampling deterministic.

During generation, we don't follow the whole chain

$$q_{\sigma, s<t}(x_s|x_t, x_0) = N\left(x_s; \sqrt{\bar{\alpha}_s}\left(\frac{x_t - \sqrt{1 - \bar{\alpha}_{t-1}}\,\epsilon_\theta^{(t)}(x_t)}{\sqrt{\bar{\alpha}_t}}\right) + \sqrt{1 - \bar{\alpha}_s - \sigma^2}\,\epsilon_\theta^{(t)}(x_t), \sigma^2 I\right)$$

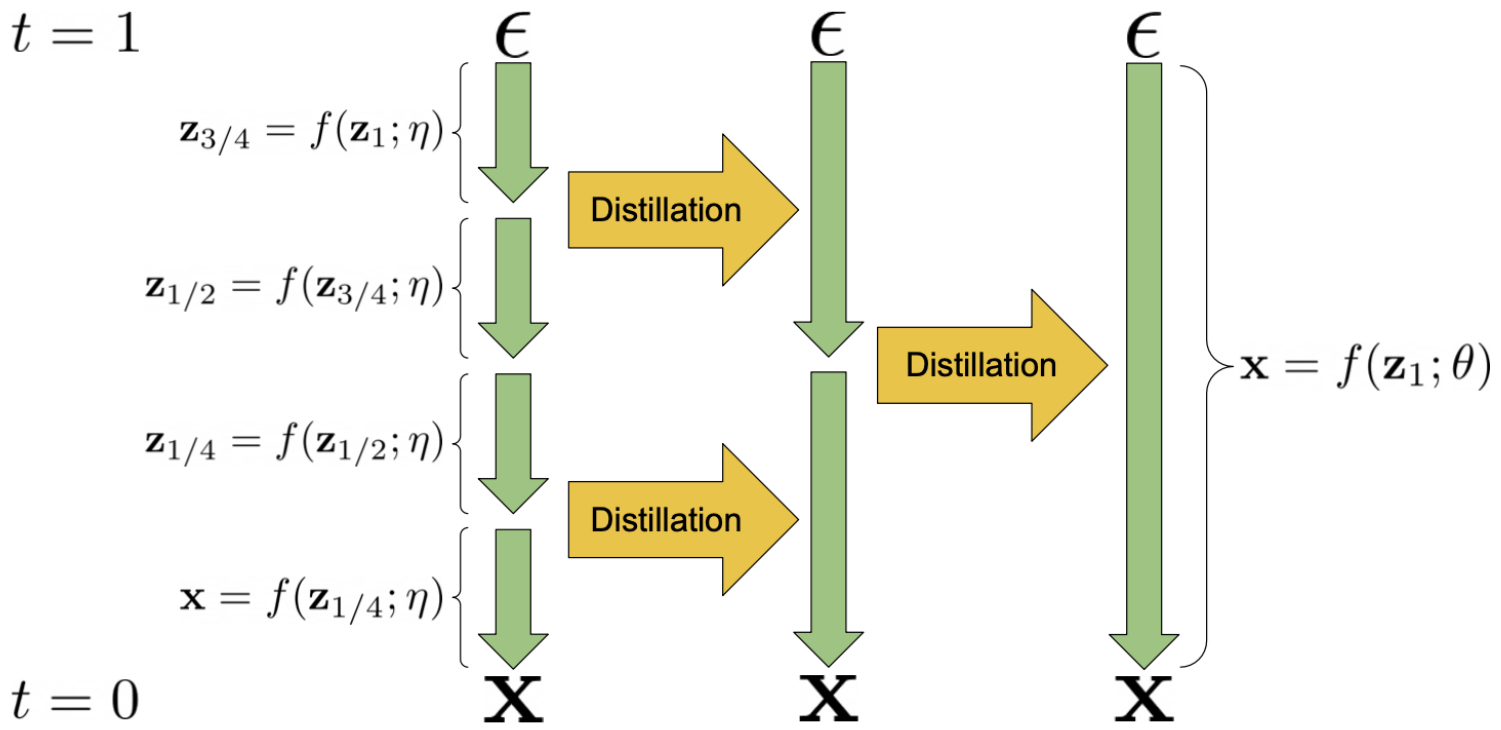
- DDIM ($\eta = 0$) can produce the best quality samples when S is small
- DDPM ($\eta = 1$) performs worse on small $S < T$, while it does perform better when we can afford to run the full reverse Markov diffusion steps ($S = T$).

Progressive Distillation

Salimans & Ho, 2022 distilled already trained deterministic samplers into new models of halved sampling steps.

Student, initialized from teacher model, denoises towards a target where one student DDIM step matches 2 teacher steps.

Every distillation iteration halves the sampling steps.



Progressive Distillation

Algorithm 1 Standard diffusion training

Require: Model $\hat{\mathbf{x}}_\theta(\mathbf{z}_t)$ to be trained
Require: Data set $\mathcal{D}$
Require: Loss weight function $w()$

while not converged **do**
  $\mathbf{x} \sim \mathcal{D}$ ▷ Sample data
  $t \sim U[0, 1]$ ▷ Sample time
  $\epsilon \sim N(0, I)$ ▷ Sample noise
  $\mathbf{z}_t = \alpha_t \mathbf{x} + \sigma_t \epsilon$ ▷ Add noise to data

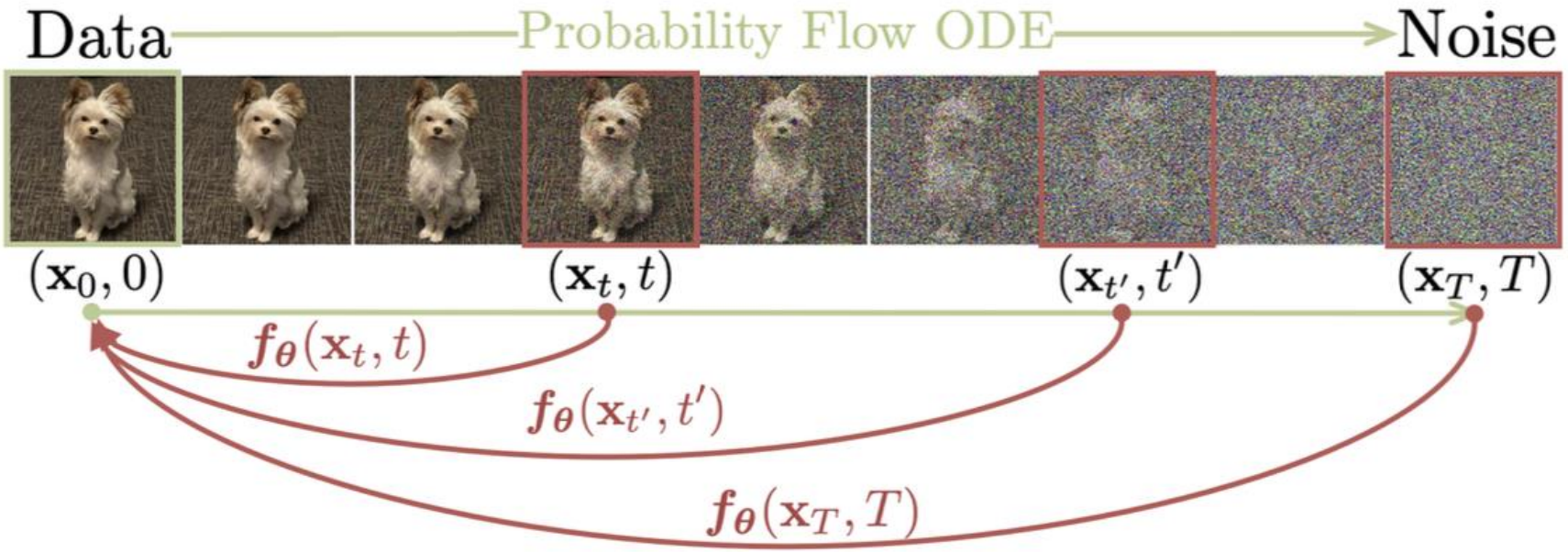
  $\tilde{\mathbf{x}} = \mathbf{x}$ ▷ Clean data is target for $\hat{\mathbf{x}}$
  $\lambda_t = \log[\alpha_t^2 / \sigma_t^2]$ ▷ log-SNR
  $L_\theta = w(\lambda_t) \|\tilde{\mathbf{x}} - \hat{\mathbf{x}}_\theta(\mathbf{z}_t)\|_2^2$ ▷ Loss
  $\theta \leftarrow \theta - \gamma \nabla_\theta L_\theta$ ▷ Optimization
end while

Algorithm 2 Progressive distillation

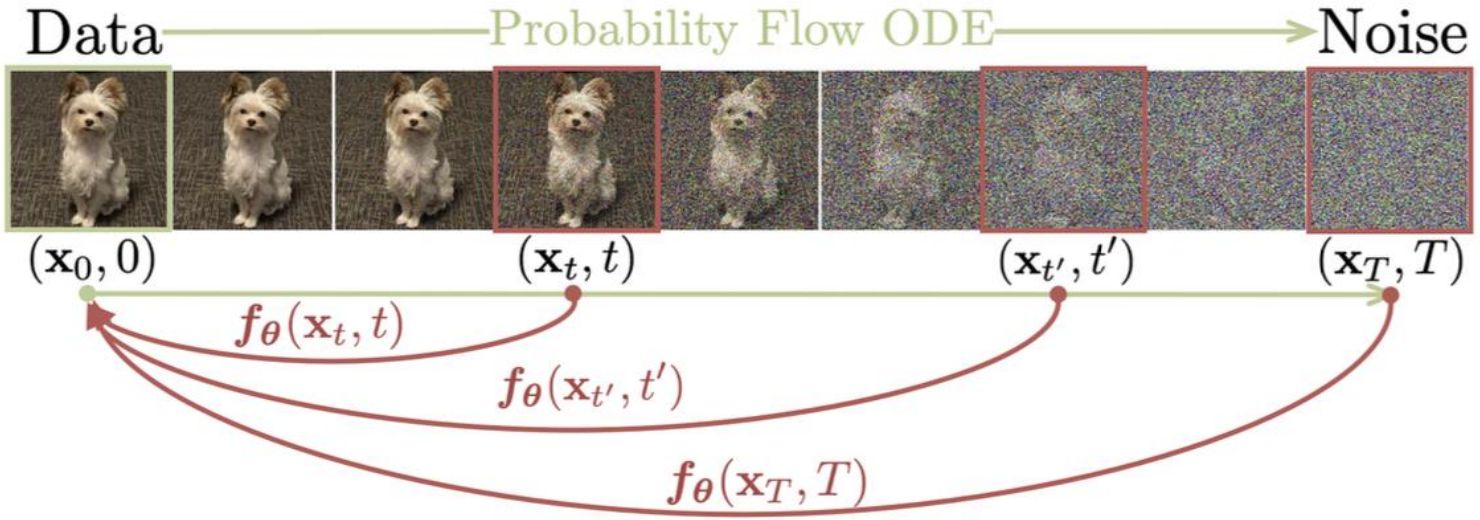
Require: Trained teacher model $\hat{\mathbf{x}}_\eta(\mathbf{z}_t)$
Require: Data set $\mathcal{D}$
Require: Loss weight function $w()$
Require: Student sampling steps N
for K iterations **do**
  $\theta \leftarrow \eta$ ▷ Init student from teacher
  while not converged **do**
    $\mathbf{x} \sim \mathcal{D}$
    $t = i/N, \;\; i \sim Cat[1, 2, \ldots, N]$
    $\epsilon \sim N(0, I)$
    $\mathbf{z}_t = \alpha_t \mathbf{x} + \sigma_t \epsilon$
    `# 2 steps of DDIM with teacher`
    $t' = t - 0.5/N, \;\; t'' = t - 1/N$
    $\mathbf{z}_{t'} = \alpha_{t'} \hat{\mathbf{x}}_\eta(\mathbf{z}_t) + \frac{\sigma_{t'}}{\sigma_t}(\mathbf{z}_t - \alpha_t \hat{\mathbf{x}}_\eta(\mathbf{z}_t))$
    $\mathbf{z}_{t''} = \alpha_{t''} \hat{\mathbf{x}}_\eta(\mathbf{z}_{t'}) + \frac{\sigma_{t''}}{\sigma_{t'}}(\mathbf{z}_{t'} - \alpha_{t'} \hat{\mathbf{x}}_\eta(\mathbf{z}_{t'}))$
    $\tilde{\mathbf{x}} = \frac{\mathbf{z}_{t''} - (\sigma_{t''}/\sigma_t)\mathbf{z}_t}{\alpha_{t''} - (\sigma_{t''}/\sigma_t)\alpha_t}$ ▷ Teacher $\hat{\mathbf{x}}$ target
    $\lambda_t = \log[\alpha_t^2 / \sigma_t^2]$
    $L_\theta = w(\lambda_t) \|\tilde{\mathbf{x}} - \hat{\mathbf{x}}_\theta(\mathbf{z}_t)\|_2^2$
    $\theta \leftarrow \theta - \gamma \nabla_\theta L_\theta$
  end while
  $\eta \leftarrow \theta$ ▷ Student becomes next teacher
  $N \leftarrow N/2$ ▷ Halve number of sampling steps
end for

Consistency Models

Song et al. 2023 learn to map intermediate noisy data points $x_t, t > 0$ on the diffusion sampling trajectory back to its origin x_0 directly. Any data points on the same trajectory is mapped to the same origin.



Consistency Models



Given a trajectory $\{x_t | t \in [\epsilon, T]\}$ *consistency function* f is defined as $f: (x_t, t) \mapsto x_\epsilon$ and the equation $f(x_t, t) = f(x_{t'}, t') = x_\epsilon$ holds true $\forall t, t' \in [\epsilon, T]$ and with $t = \epsilon$, f is the identify function.

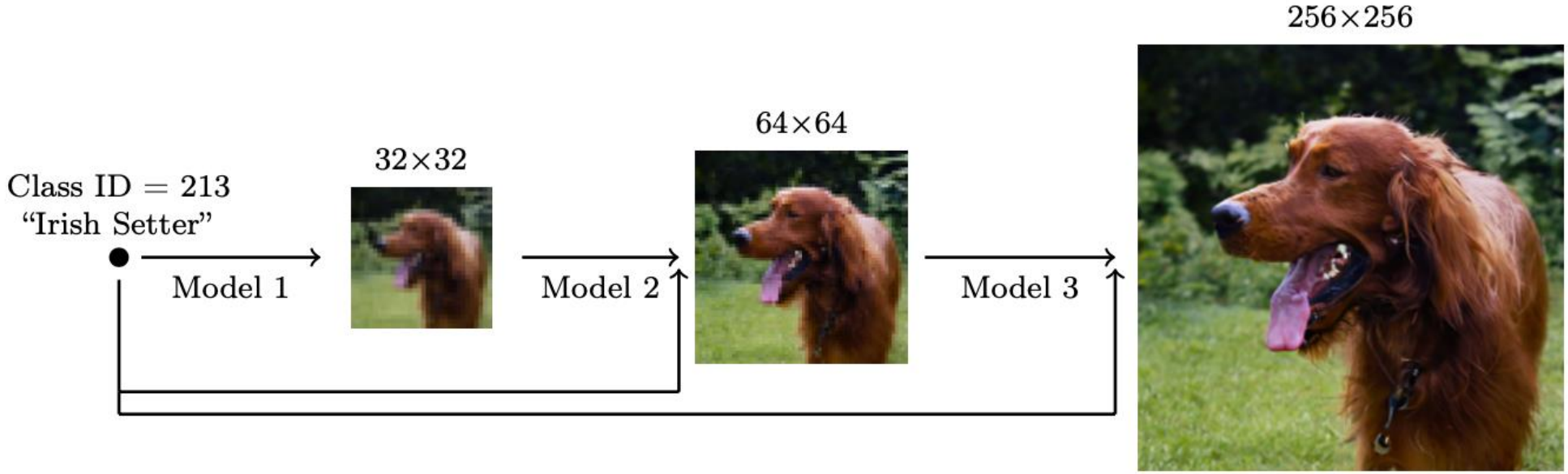
Consistency models generate samples in a single step, trading computation for better quality following a multi-step sampling process.

Scaling Up Resolution and Quality

Ho et al. (2021) propose diffusion models at increasing resolutions.

Noise conditioning augmentation between pipeline models is used:

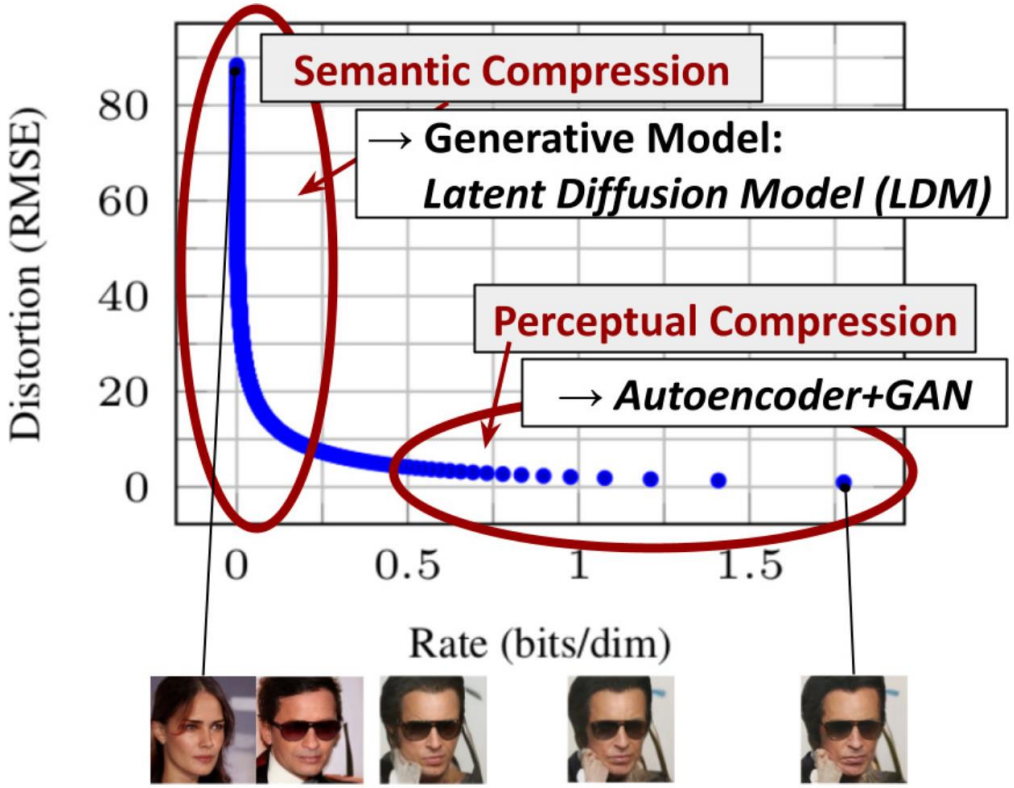
- Apply strong data augmentation to the conditioning input z of each super-resolution model $p_\theta(x|z)$.
- Conditioning noise helps reduce compounding error in the pipeline setup.



Latent Diffusion Models

Most bits of an image contribute to perceptual details and the semantic and conceptual composition still remains after aggressive compression.

LDM decomposes the perceptual and semantic compressions by first trimming off pixel-level redundancy with autoencoder then manipulating semantic concepts with DDPM



Latent diffusion model runs the diffusion process in the latent space.

Latent Diffusion Model

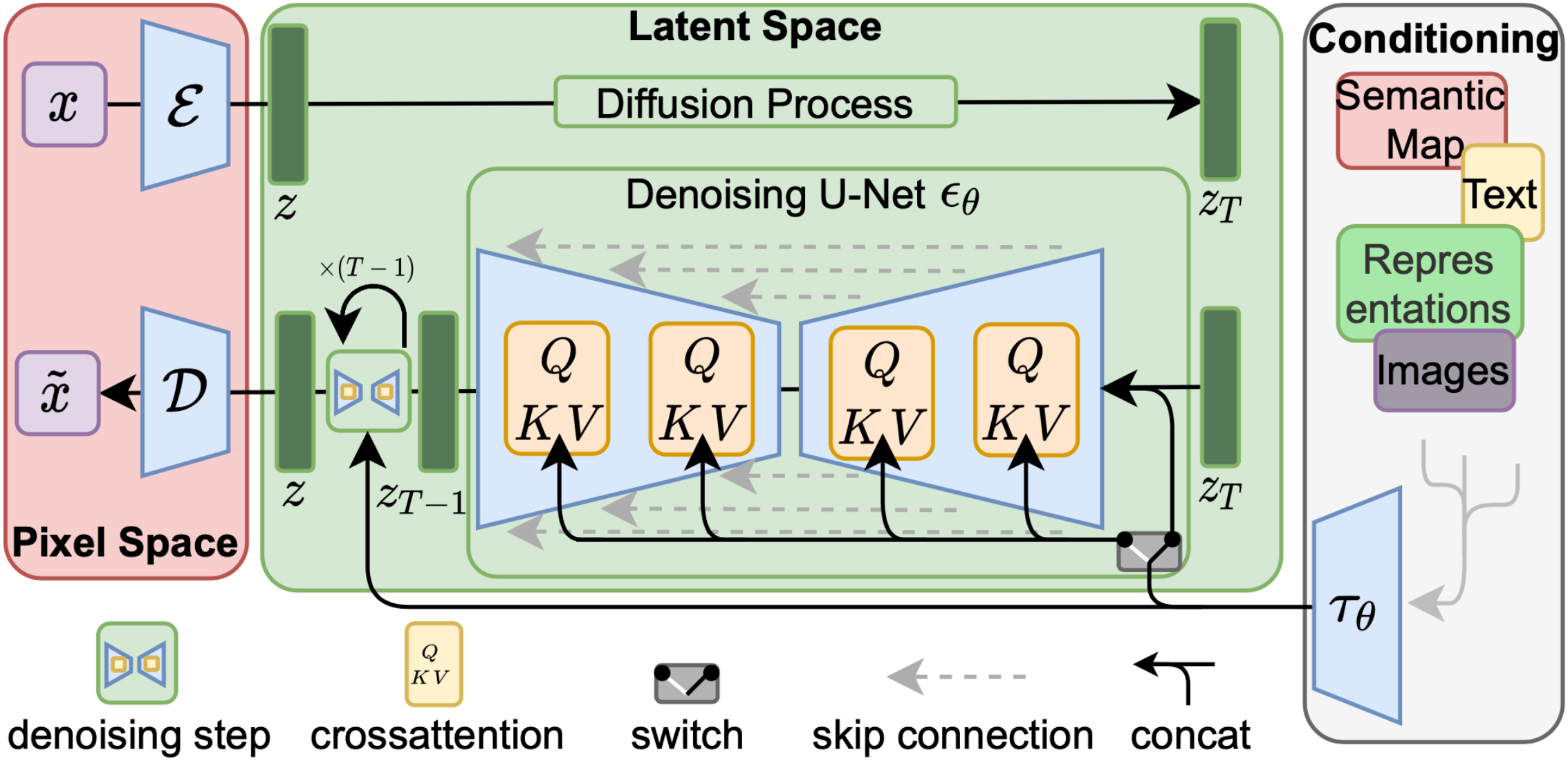
An encoder E is used to compress the input image $x \in R^{H \times W \times 3}$ to a smaller 2D latent vector $z = E(x) \in R^{h \times w \times c}$, with downsampling rate $f = H/h = W/w = 2^m, m \in N$.

A decoder D reconstructs images from the latent vector, $\tilde{x} = D(z)$.

Rombach & Blattmann, et al. (2022) explored two types of regularization in autoencoder training to avoid arbitrarily high-variance in latent spaces

- *KL-reg*: A small KL penalty towards a standard normal distribution over the learned latent, similar to VAE.
- *VQ-reg*: Uses a vector quantization layer within the decoder, like VQVAE but the quantization layer is absorbed by the decoder.

Latent Diffusion Model



Acknowledgements

Lil'Log | **What are Diffusion Models?**

Date: July 11, 2021 | Estimated Reading Time: 32 min | Author: Lilian Weng

https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

Kemal Erdem, (Nov 2023). "Step by Step visual introduction to Diffusion Models.".
https://erdem.pl/2023/11/step-by-step-visual-introduction-to-diffusion-models

Step by step implementation of a diffusion model

- https://huggingface.co/blog/annotated-diffusion

Other useful online resources

- https://theaisummer.com/diffusion-models/
- https://learnopencv.com/image-generation-using-diffusion-models/
- https://developer.nvidia.com/blog/improving-diffusion-models-as-an-alternative-to-gans-part-1/

Evaluation of Day 3 Lecture

https://forms.office.com/e/bwuwAijSzn

- Now it is closed, I will open it at the end
- You have 20 minutes to complete
- There are 32 questions 1 point each
- Each question 4 choices only 1 correct
- Best and worst (at class level) removed

POLITECNICO
MILANO 1863

Advanced Deep Learning

- Image Generation with DDPM -

Matteo Matteucci, PhD (matteo.matteucci@polimi.it)
Artificial Intelligence and Robotics Laboratory
Politecnico di Milano

AIRLAB
ARTIFICIAL INTELLIGENCE AND ROBOTICS LAB