Practices in visual computing 2

Lab5: Diffusion Models

Simon Fraser University
Spring 2025

Generative Models

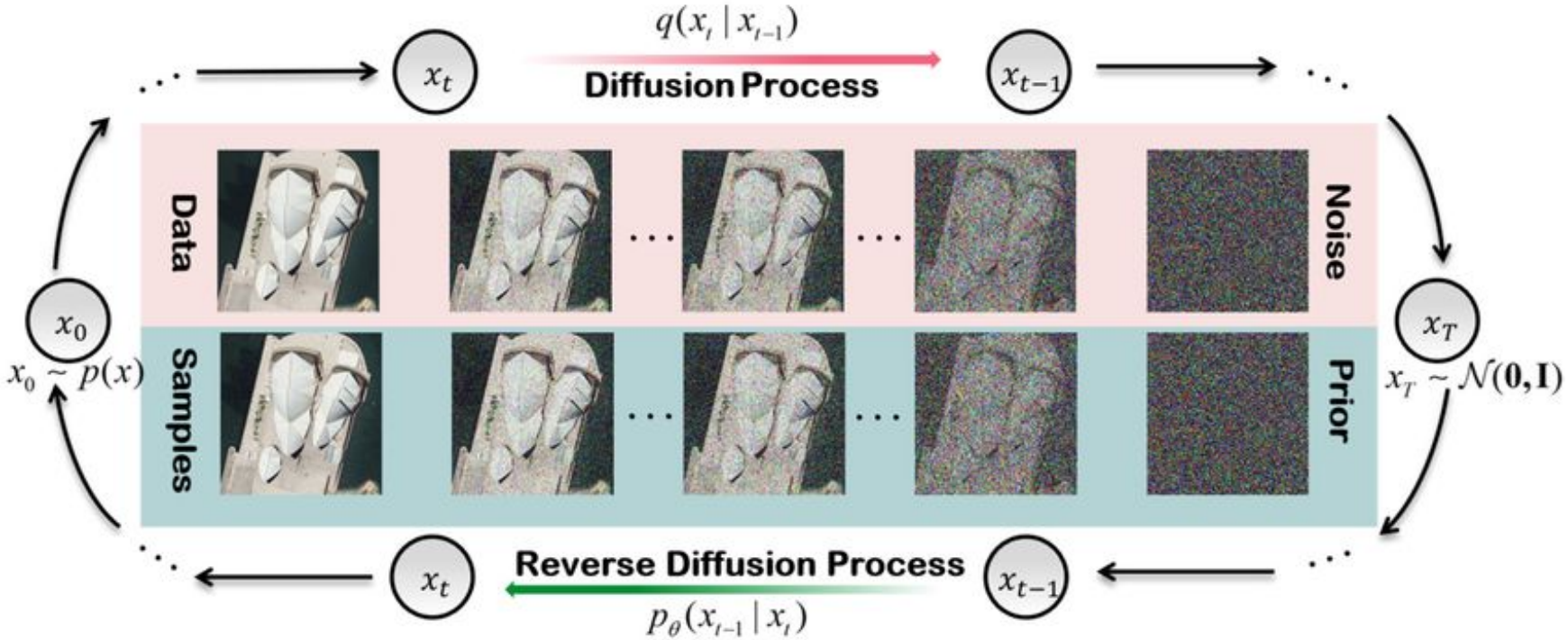
Models that learn the underlying distribution of data so they can generate new data samples that resemble the training data.

Examples of Generative Models:

- Variational Autoencoders (VAEs)
- Generative Adversarial Networks (GANs)
- Flow-based Models
- **Diffusion Models**

What Is a Diffusion Process?

A process that gradually adds noise to data until all structure is lost (like repeatedly blurring an image).



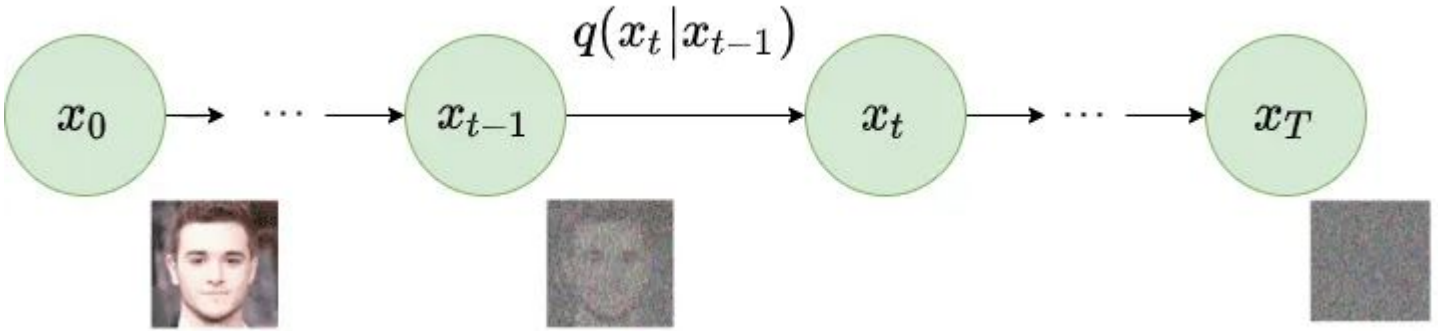
Forward Diffusion Process

Start with real data x_0.

At each step t, add a small amount of noise, getting x_t.

After T steps, the data is almost pure noise.

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



The Reparameterization Trick

Start with real data x_0.

At each step t, add a small amount of noise, getting x_t.

After T steps, the data is almost pure noise.

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

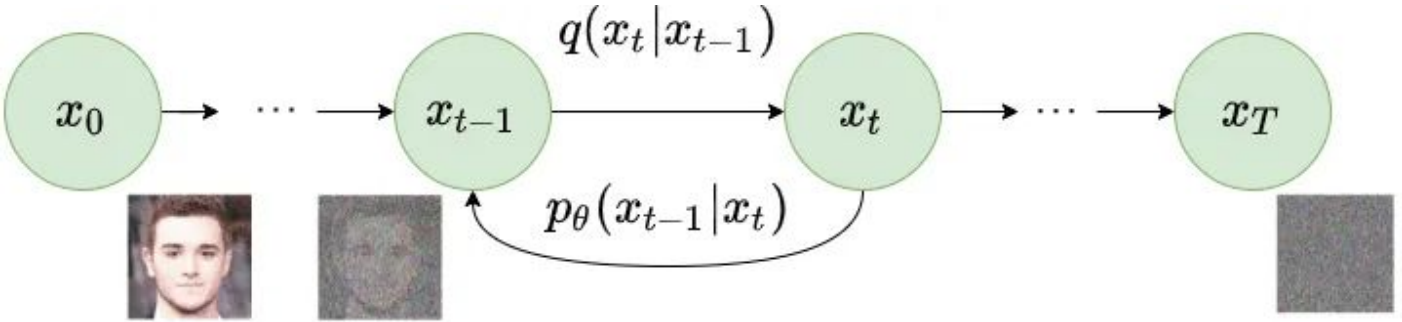
$$\mathbf{x}_t \sim q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I})$$

Reverse Denoising

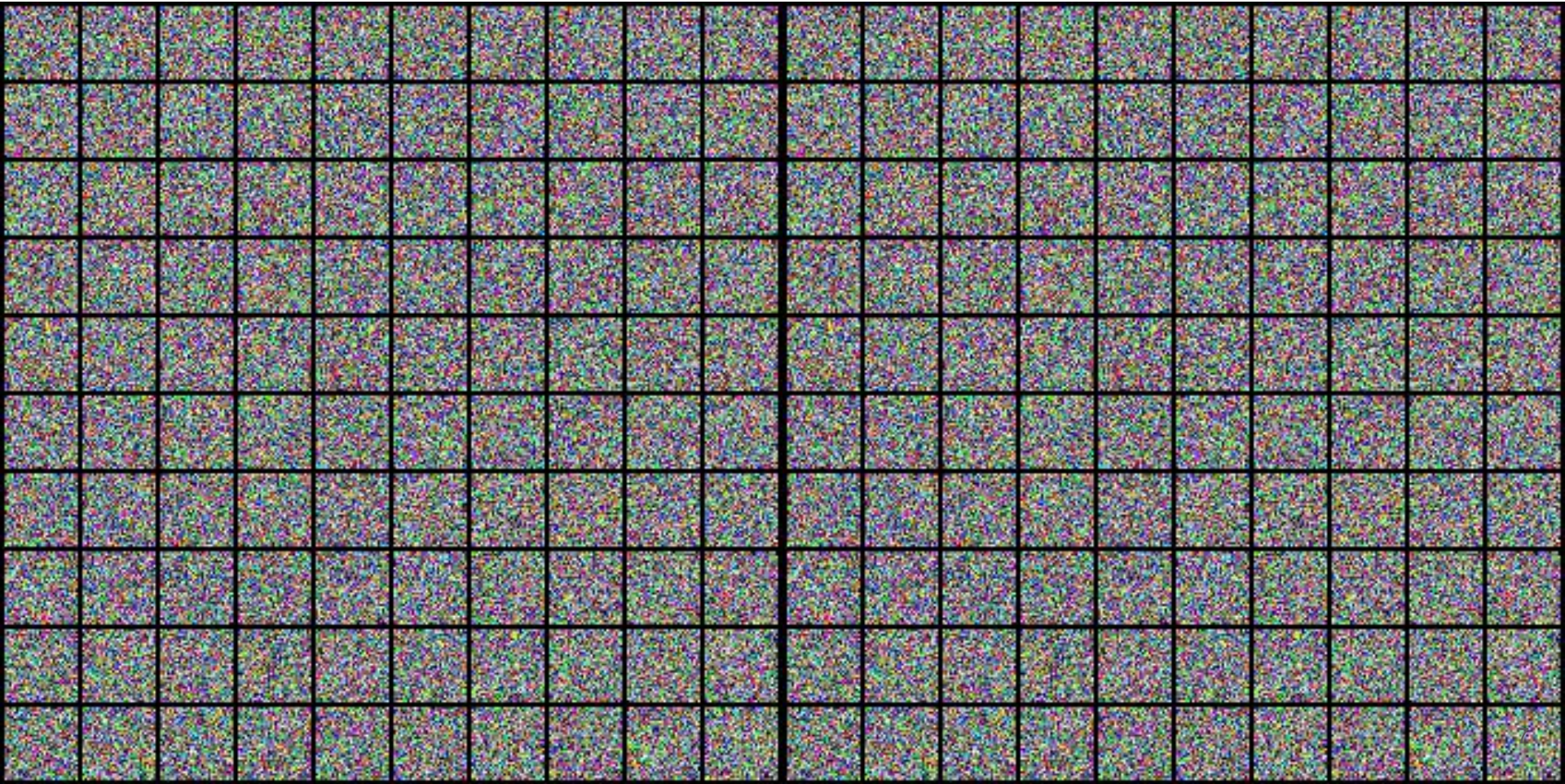
Start with random noise x_T.

Iteratively denoise from T down to 0, gradually “removing noise” and reconstructing a sample that looks like real data.

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



Examples



Diffusion Models Beat GANs on Image Synthesis

BigGAN-deep | Diffusion model | Training set

Training

Learn reverse Markov transitions to maximize training data likelihood.

Minimize the variational upper bound on negative log-likelihood.

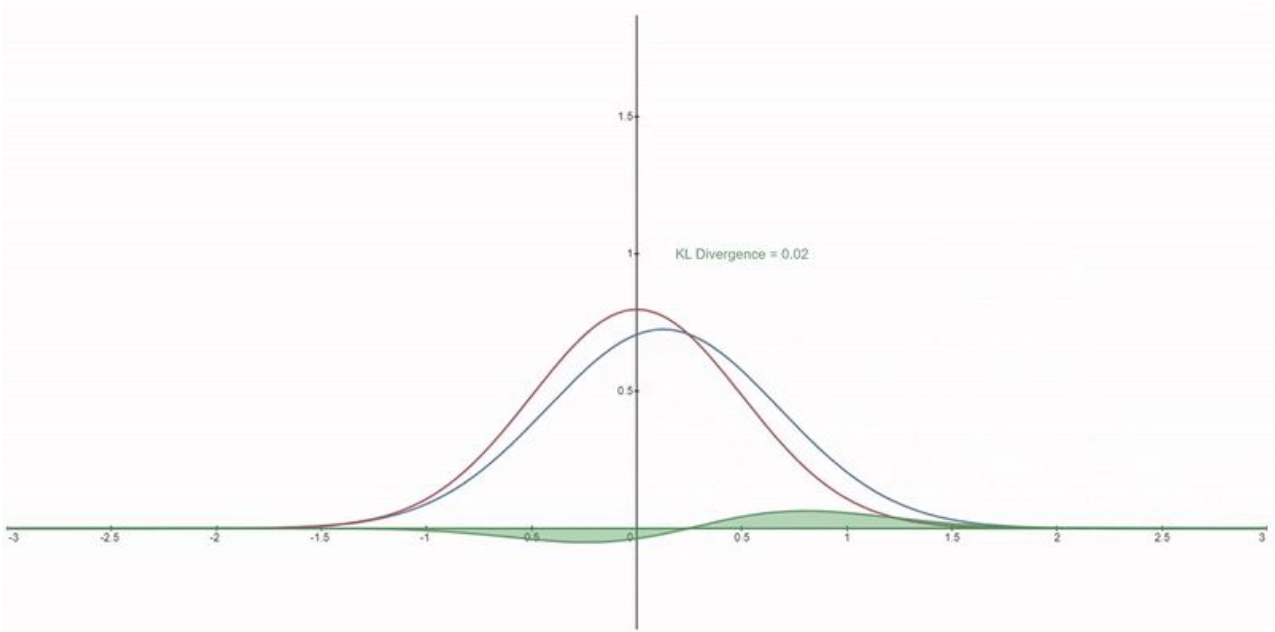
Why? Direct likelihood computation is intractable.

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

Kullback-Leibler (KL) Divergences.

Measures how one probability distribution differs from another.

$$D_{\mathrm{KL}}(P \parallel Q) = \int_{-\infty}^{\infty} p(x) \log\left(\frac{p(x)}{q(x)}\right) dx$$



Casting vlb in Terms of KL Divergences

Replaced the original objective function with vlb.

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

vlb can be expressed in terms of KL Divergences

$$L_{vlb} = L_0 + L_1 + ... + L_{T-1} + L_T$$

$$L_0 = -\log p_\theta(x_0|x_1)$$

$$L_{t-1} = D_{KL}(q(x_{t-1}|x_t, x_0) \,||\, p_\theta(x_{t-1}|x_t))$$

$$L_T = D_{KL}(q(x_T|x_0) \,||\, p(x_T))$$

Reverse Process and $L_{1:T-1}$

We choose reverse Markov transitions as a Gaussian

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) := \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t))$$

For simplicity, we also set

$$\boldsymbol{\Sigma}_\theta(x_t, t) = \sigma_t^2 \mathbb{I}$$
$$\sigma_t^2 = \beta_t$$

So, we have

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

Casting vlb in Terms of KL Divergences

We have

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

Which allows us to transform

$$L_{t-1} = D_{KL}(q(x_{t-1}|x_t, x_0) \,||\, p_\theta(x_{t-1}|x_t))$$

To

$$L_{t-1} \propto ||\tilde{\mu}_t(x_t, x_0) - \mu_\theta(x_t, t)||^2$$

Reparameterization

So we have

$$L_{t-1} \propto ||\tilde{\mu}_t(x_t, x_0) - \mu_\theta(x_t, t)||^2$$

Authors of “Denoising Diffusion Probabilistic Models” paper found learning noise yields better results.

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

Where

$$\alpha_t := 1 - \beta_t \quad \text{and} \quad \bar{\alpha}_t := \prod_{s=1}^{t} \alpha_s$$

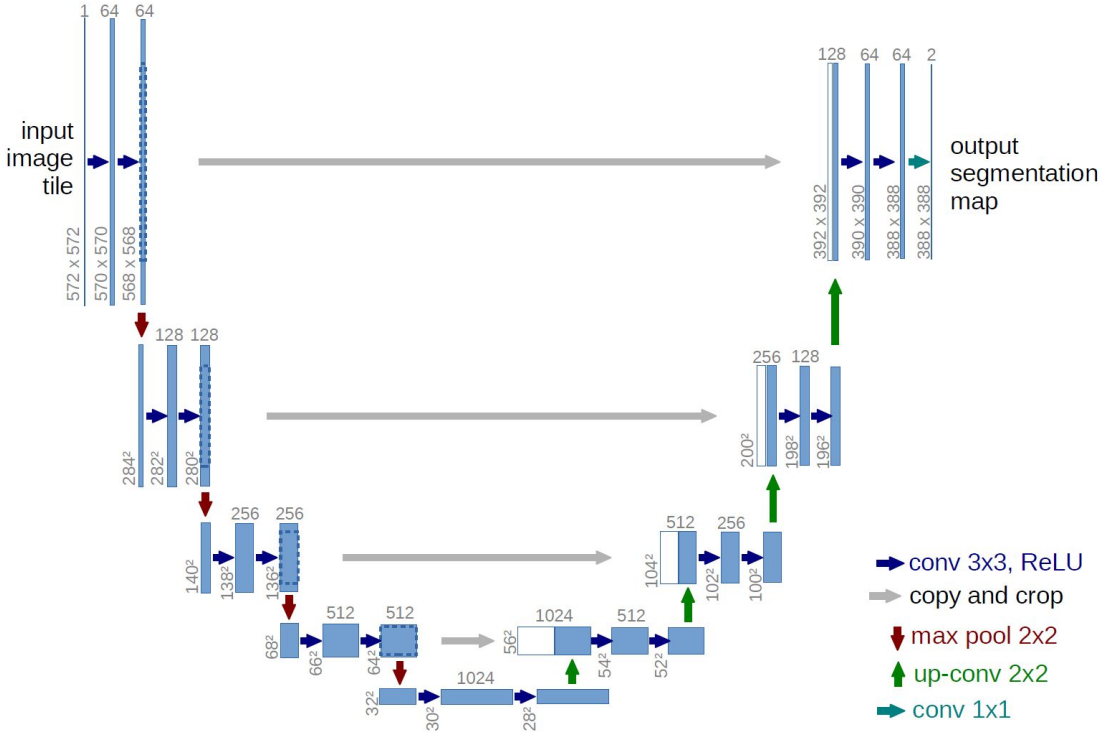
Result in a new objective function

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

Common Architectures

U-Net-based models (popular for image diffusion).

Transformer-based models (used for scalability and flexibility).



Training and Sampling

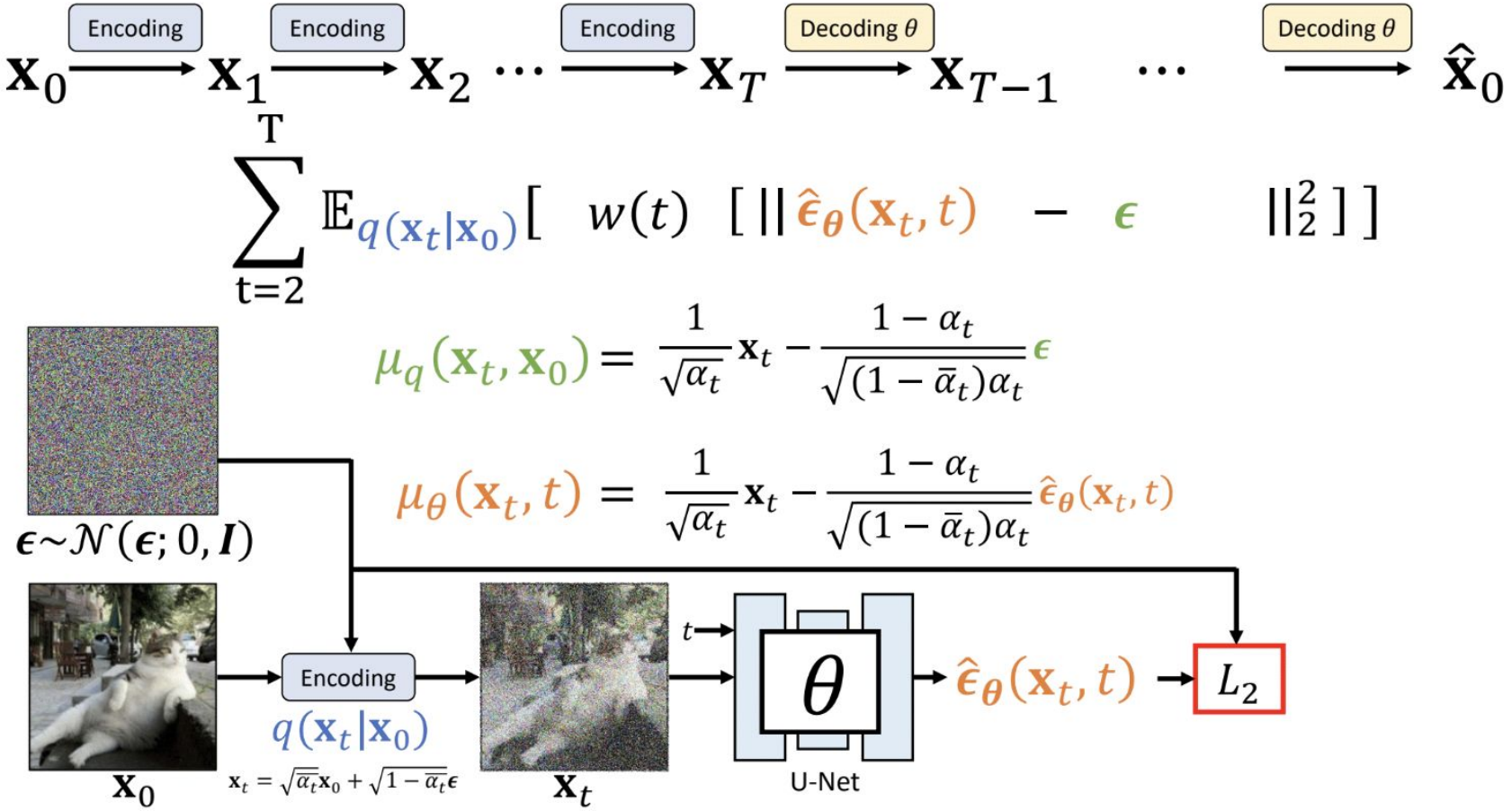
Algorithm 1 Training

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** $\mathbf{x}_0$

Training


Encoding
Encoding
Encoding
Decoding θ
Decoding θ
$\mathbf{x}_0$ $\mathbf{x}_1$ $\mathbf{x}_2$ $\cdots$ $\mathbf{x}_T$ $\mathbf{x}_{T-1}$ $\cdots$ $\hat{\mathbf{x}}_0$
$\sum_{t=2}^{T} \mathbb{E}_{q(\mathbf{x}_t|\mathbf{x}_0)}[\ w(t)\ [\ ||\hat{\epsilon}_\theta(\mathbf{x}_t, t) - \epsilon\ ||_2^2\ ]\ ]$
$\mu_q(\mathbf{x}_t, \mathbf{x}_0) = \frac{1}{\sqrt{\alpha_t}}\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{(1-\bar{\alpha}_t)\alpha_t}}\epsilon$
$\mu_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{\alpha_t}}\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{(1-\bar{\alpha}_t)\alpha_t}}\hat{\epsilon}_\theta(\mathbf{x}_t, t)$
$\epsilon \sim \mathcal{N}(\epsilon; 0, I)$
Encoding
$q(\mathbf{x}_t|\mathbf{x}_0)$
$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$
$\mathbf{x}_0$
$\mathbf{x}_t$
t
θ
U-Net
$\hat{\epsilon}_\theta(\mathbf{x}_t, t)$
L_2

Scheduler

A scheduler in diffusion models controls how much noise is added at each time step during the forward diffusion process (and sometimes how noise is removed during reverse steps).

It typically defines a sequence of β_t parameters indicating the noise variance at time t.

Common Noise Schedules

Linear Schedule

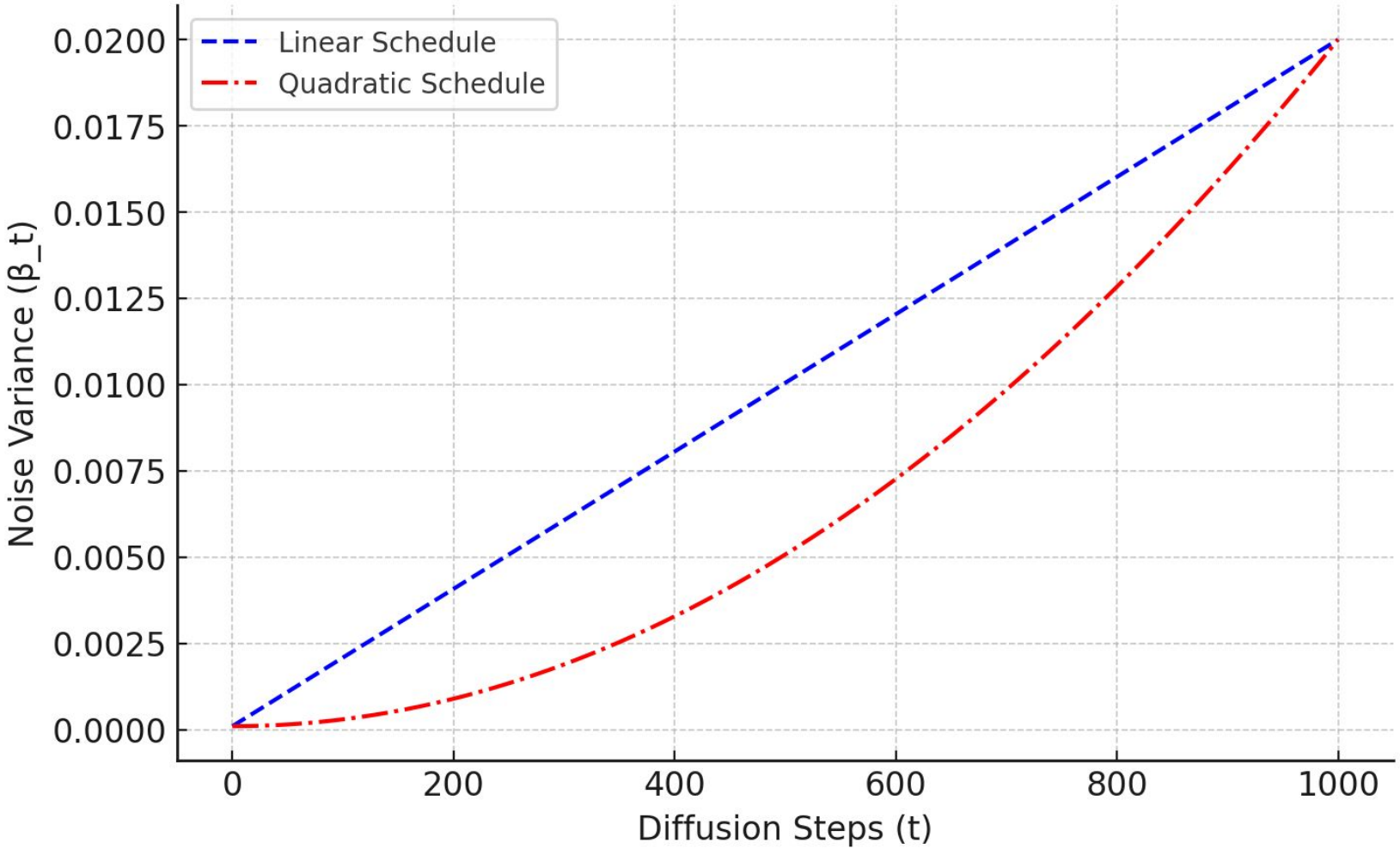
β_t (noise variance) increases linearly from a small value β_{min} to a larger value β_{max}.

$$\beta_t = \beta_1 + \frac{t-1}{T-1}(\beta_T - \beta_1), \quad t = 1, \ldots, T$$

Quadratic Schedule

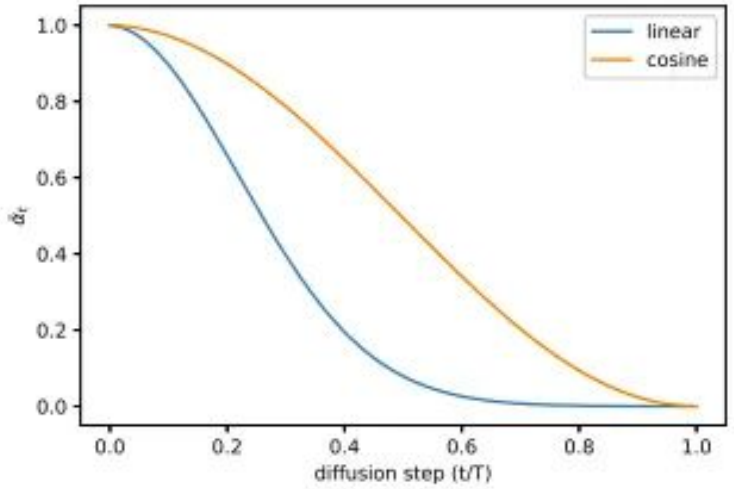
βt (noise variance) follows a quadratic curve:

$$\beta_t = \left(\sqrt{\beta_1} + \frac{t-1}{T-1}(\sqrt{\beta_T} - \sqrt{\beta_1}) \right)^2, \quad t = 1, \ldots, T$$


Comparison of Linear vs Quadratic Noise Schedules
Linear Schedule
Quadratic Schedule
Noise Variance (β_t)
0.0200
0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
0
200
400
600
800
1000
Diffusion Steps (t)

Other Noise Schedules

Schedule	Definition ($i = \frac{t-1}{T-1}$)	SNR(T)	$\sqrt{\bar{\alpha}_T}$
Linear [3]	$\beta_t = 0.0001 \cdot (1-i) + 0.02 \cdot i$	4.035993e-05	0.006353
Cosine [8]	$\beta_t = \min(1 - \frac{\bar{\alpha}_t}{\bar{\alpha}_{t-1}}, 0.999), \bar{\alpha}_t = \frac{f(t)}{f(0)}, f(t) = \cos(\frac{i+0.008}{1+0.008} \cdot \frac{\pi}{2})^2$	2.428735e-09	4.928220e-05
Stable Diffusion [10]	$\beta_t = (\sqrt{0.00085} \cdot (1-i) + \sqrt{0.012} \cdot i)^2$	0.004682	0.068265



Sinusoidal Embeddings

In diffusion models, each timestep t needs to be encoded into a meaningful representation for the neural network.

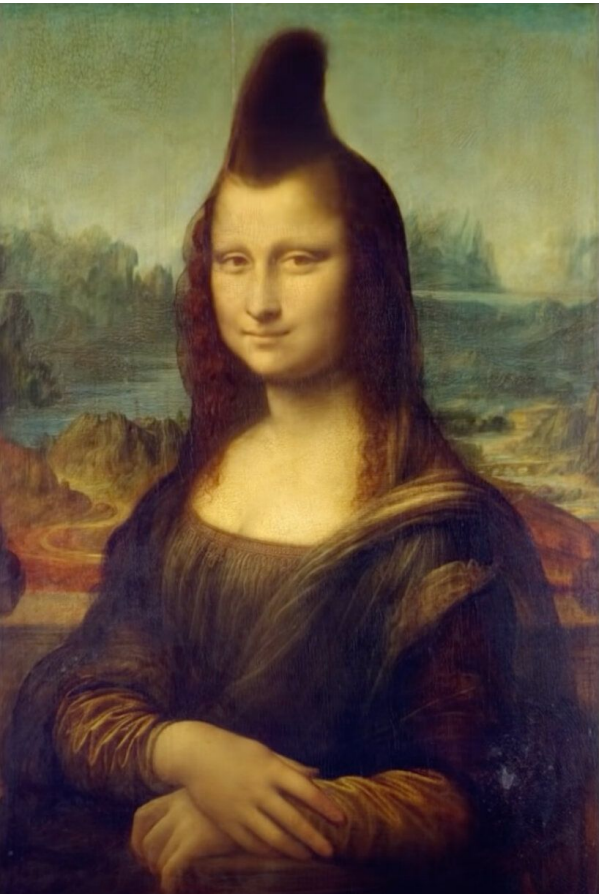
Instead of using raw scalar values t, sinusoidal embeddings (also used in Transformers) provide a more structured way to encode time information.

$$\text{emb}(t) = \left[\cos(t \cdot \omega_1),\ \sin(t \cdot \omega_1),\ \ldots,\ \cos(t \cdot \omega_{n/2}),\ \sin(t \cdot \omega_{n/2})\right]$$

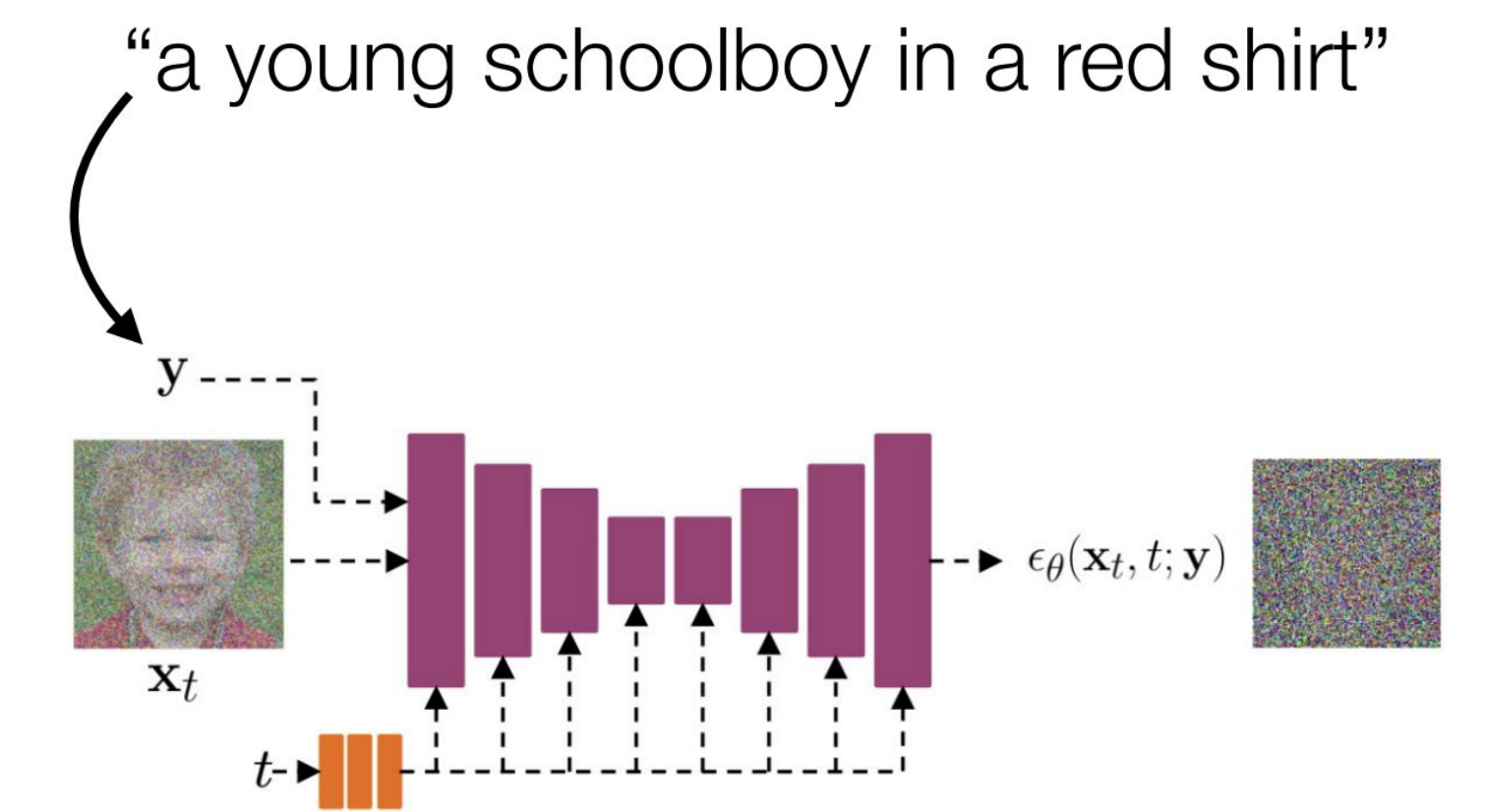
where

$$\omega_i = \exp\left(-\frac{\log(\texttt{max\_period})}{n} \times i\right), \quad i \in [1, \ldots, n/2]$$

DALL·E 2



Explicit Conditioning



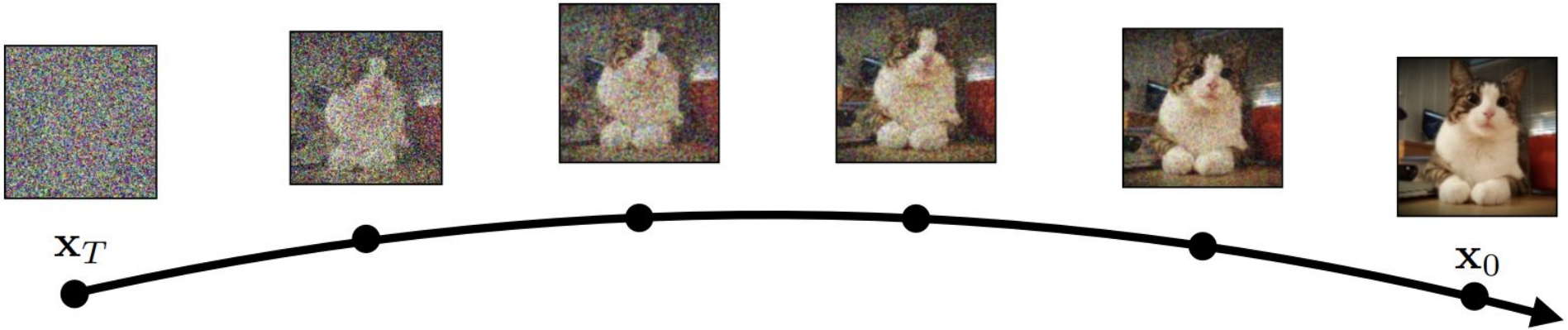
Classifier Guidance

Modify the sampling process in a diffusion model so that generated samples align with a given class or condition.

Idea: Use a separate classifier $p_\phi(y|x_t)$ (where y is the label or condition) to adjust the reverse diffusion steps.

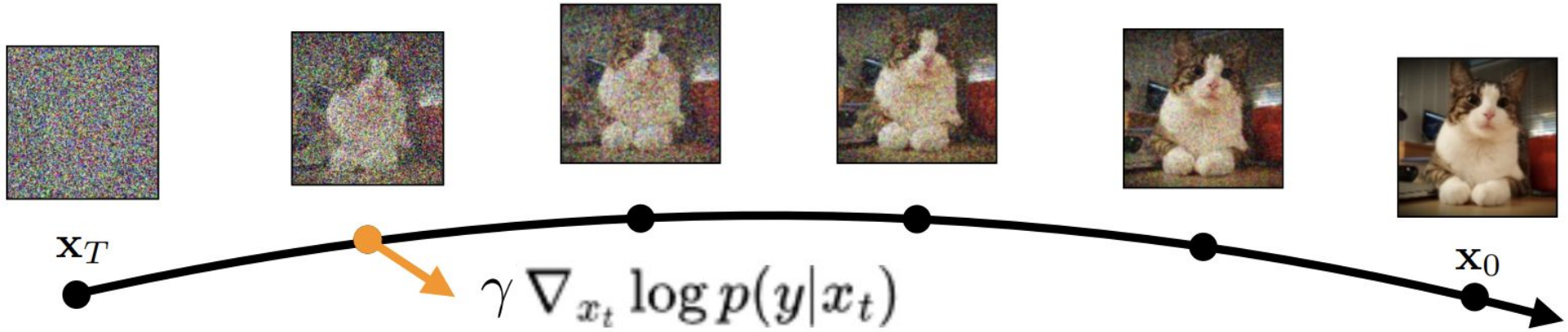
Reverse Diffusion Steps

Diffusion goes from noise to real images step-by-step.



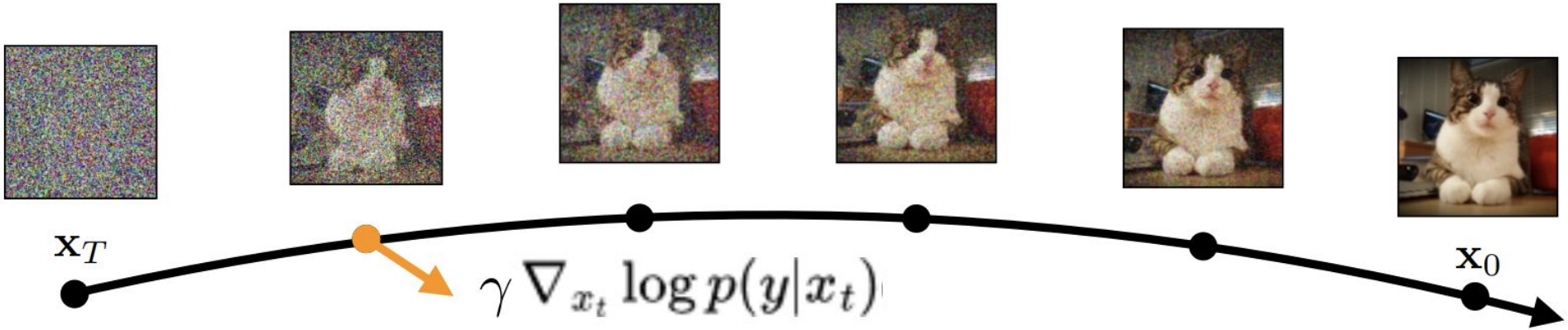
Classifier Guidance

Perturb using gradients of a classifier:



Classifier Guidance

Perturb using gradients of a classifier:



$$\tilde{\epsilon}_t(x_t, t, y) = \epsilon_\theta(x_t, t) - \gamma \nabla_{x_t} \log p(y|x_t)$$

Classifier Guidance

Pros

- Strong Control: We can push the generative model toward a label.
- Separation of concerns: The classifier and diffuser can be trained differently or updated independently.

Cons

- Time-consuming: Requires a separate classifier that must be trained on noisy samples at each diffusion step.
- Complex pipeline: Two models (diffusion + classifier) must remain in sync.

Classifier Free Guidance

Motivation:

- Avoid the need to train and maintain a separate classifier.
- Achieve conditional generation with a single diffusion model.

Idea:

- Let the diffusion model itself learn both an unconditional and conditional denoising function.
- Combine them at inference to guide samples without an external classifier.

Classifier Free Guidance

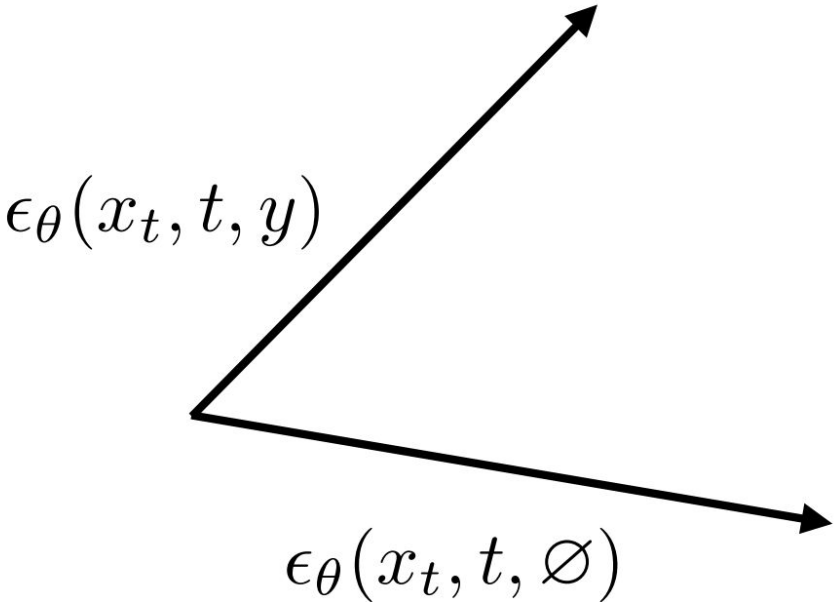
Train one model to produce:

- Conditional prediction: $\epsilon_\theta(x_t, t, y)$
- Unconditional prediction: $\epsilon_\theta(x_t, t, \varnothing)$

Classifier Free Guidance

Train one model to produce:

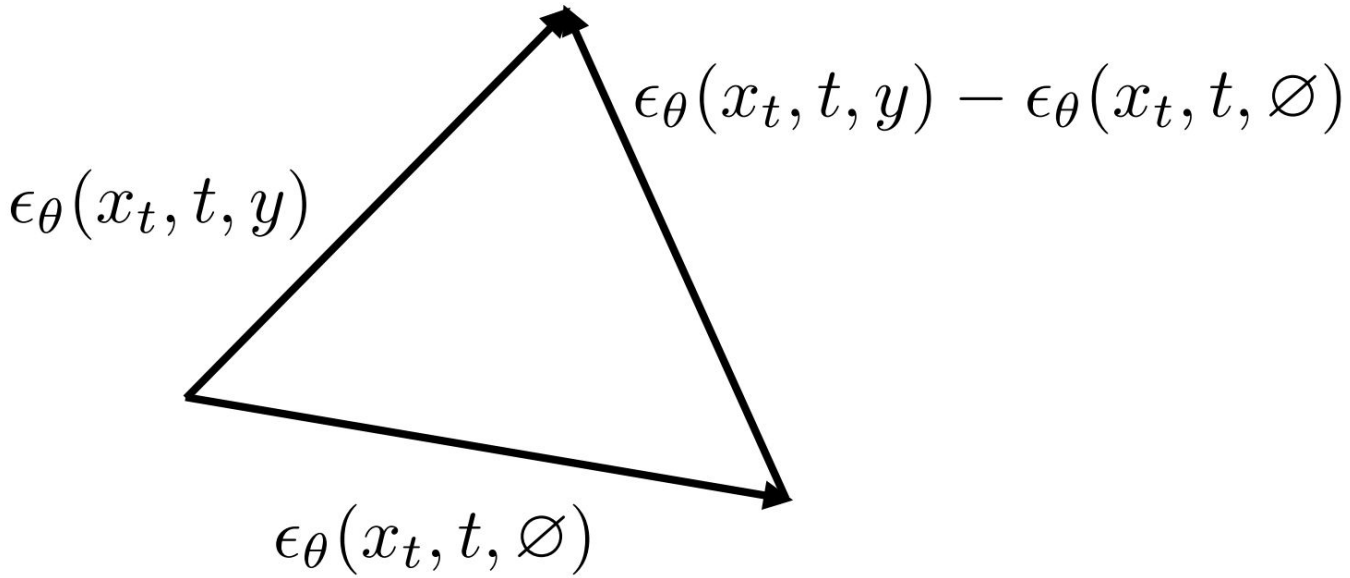
- Conditional prediction: $\epsilon_\theta(x_t, t, y)$
- Unconditional prediction: $\epsilon_\theta(x_t, t, \varnothing)$



Classifier Free Guidance

Train one model to produce:

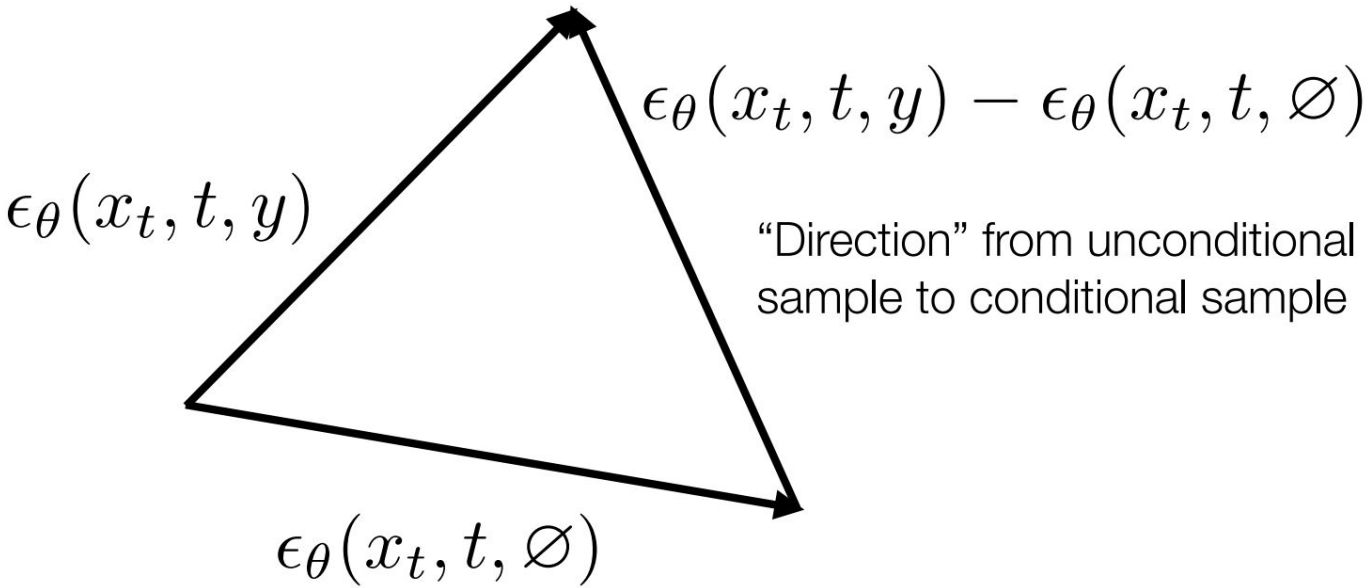
- Conditional prediction: $\epsilon_\theta(x_t, t, y)$
- Unconditional prediction: $\epsilon_\theta(x_t, t, \varnothing)$



Classifier Free Guidance

Train one model to produce:

- Conditional prediction: $\epsilon_\theta(x_t, t, y)$
- Unconditional prediction: $\epsilon_\theta(x_t, t, \varnothing)$



Classifier Free Guidance

$$\tilde{\epsilon}(x_t, t, y) = \epsilon_\theta(x_t, t, \varnothing) + \gamma(\epsilon_\theta(x_t, t, y) - \epsilon_\theta(x_t, t, \varnothing))$$

"Direction" from unconditional to conditional

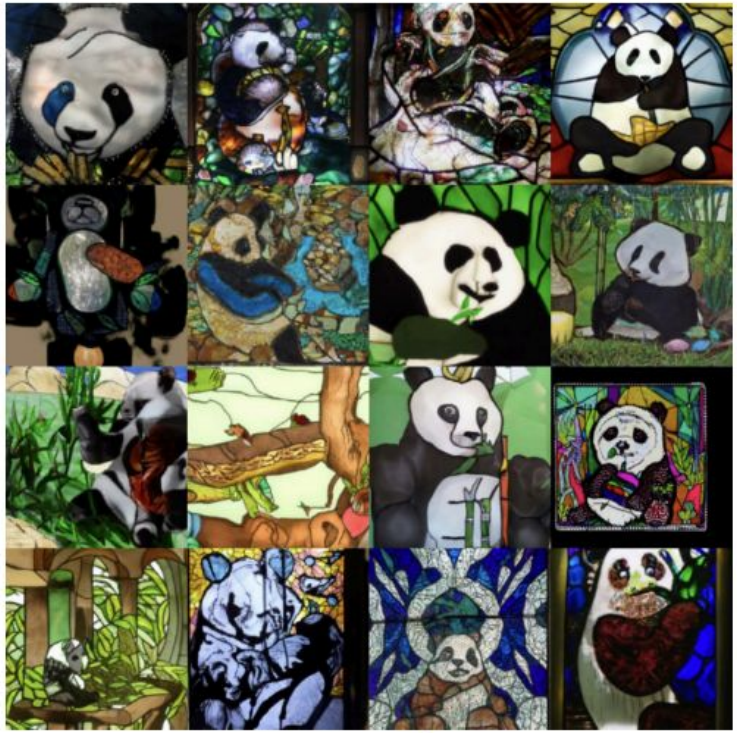
Classifier Free Guidance

"A stained glass window of a panda eating bamboo"

$\gamma = 1$

Classifier Free Guidance

"A stained glass window of a panda eating bamboo"

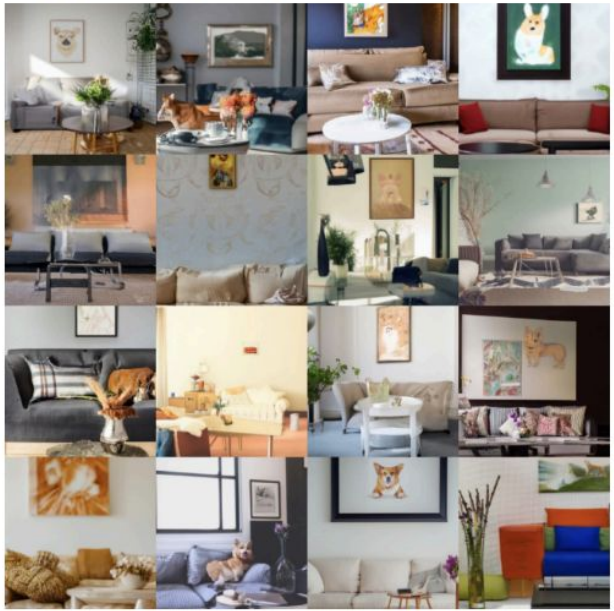


$\gamma = 1$

$\gamma = 3$

Classifier Free Guidance

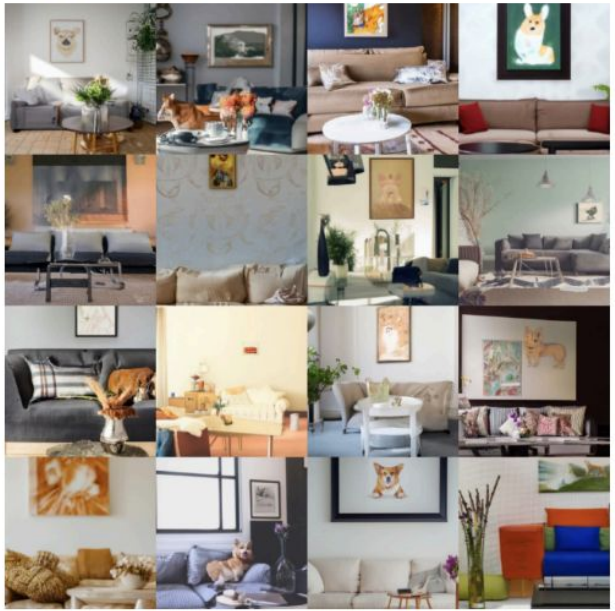
"A cozy living room with a painting of a corgi on the wall above a couch and a round coffee table in front of a couch and a vase of flowers on a coffee table."



$\gamma = 1$

Classifier Free Guidance

"A cozy living room with a painting of a corgi on the wall above a couch and a round coffee table in front of a couch and a vase of flowers on a coffee table."



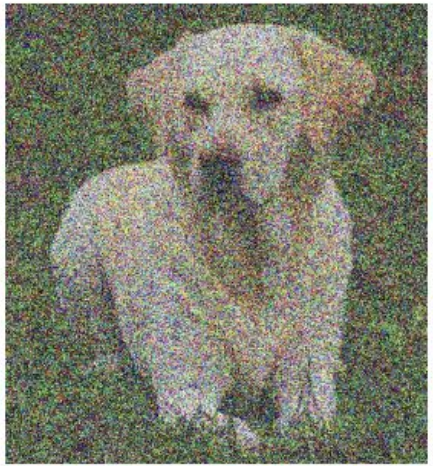
$\gamma = 1$

$\gamma = 3$

Image Editing with Diffusion Models

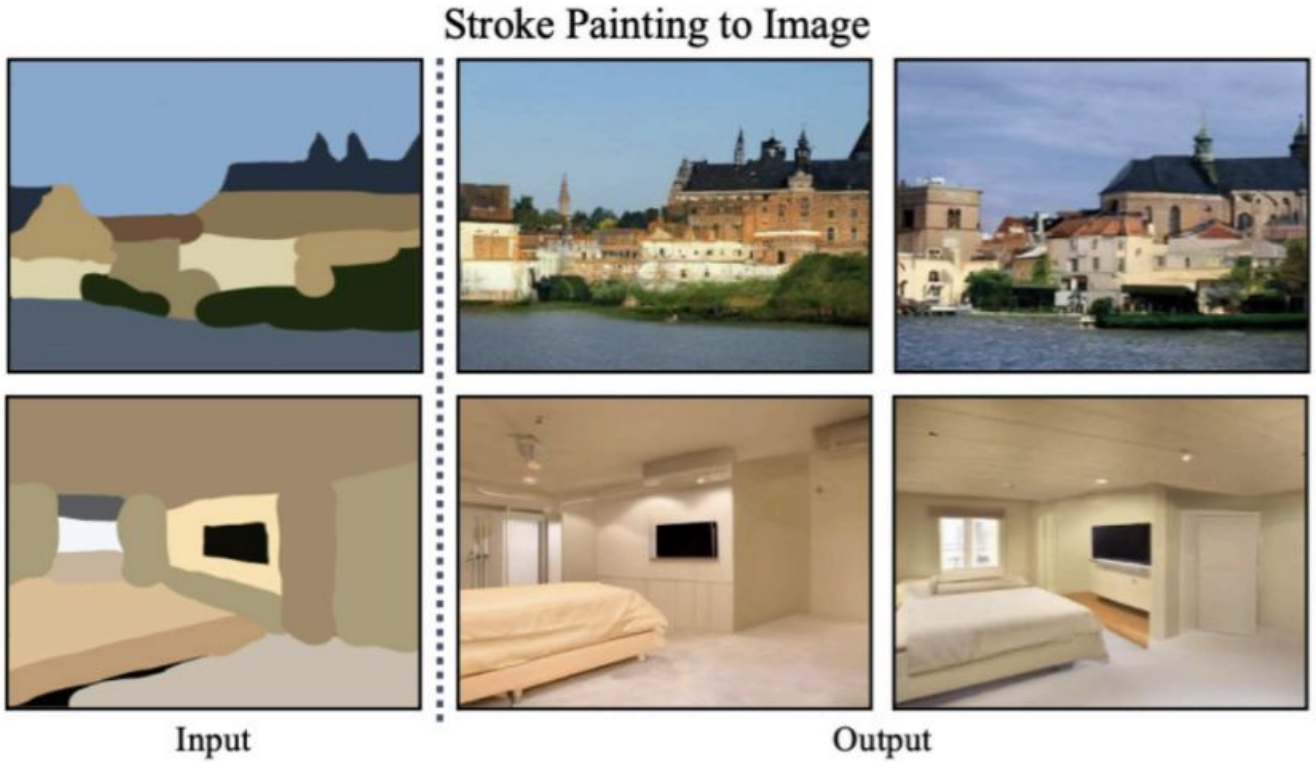
SDEdit:

- Add noise to an image, and then remove it with a diffusion model.



Add noise by running the forward process $q(x_t|x_0)$

SDEdit

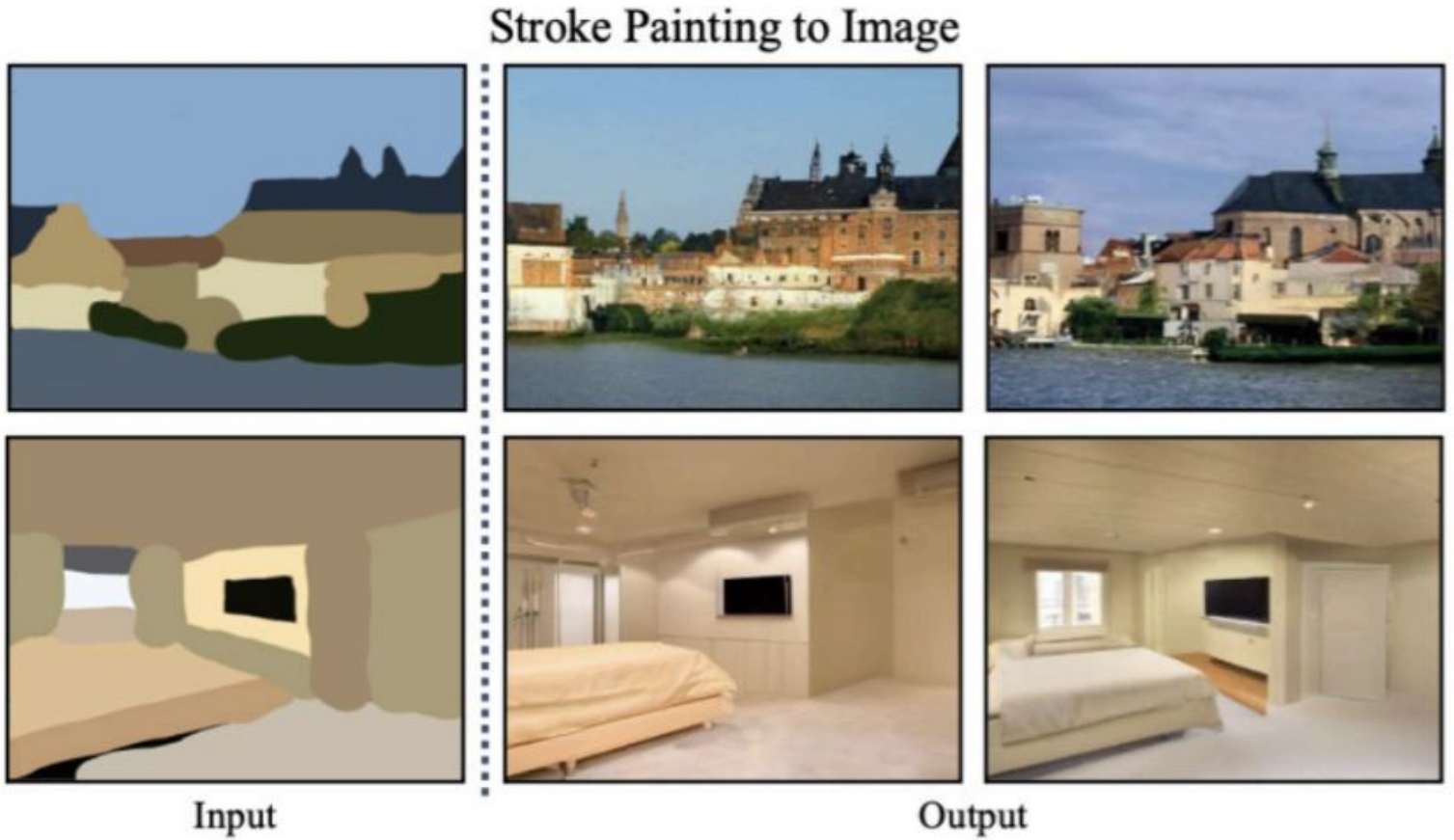


SDEdit

Stroke Painting to Image

Input

SDEdit

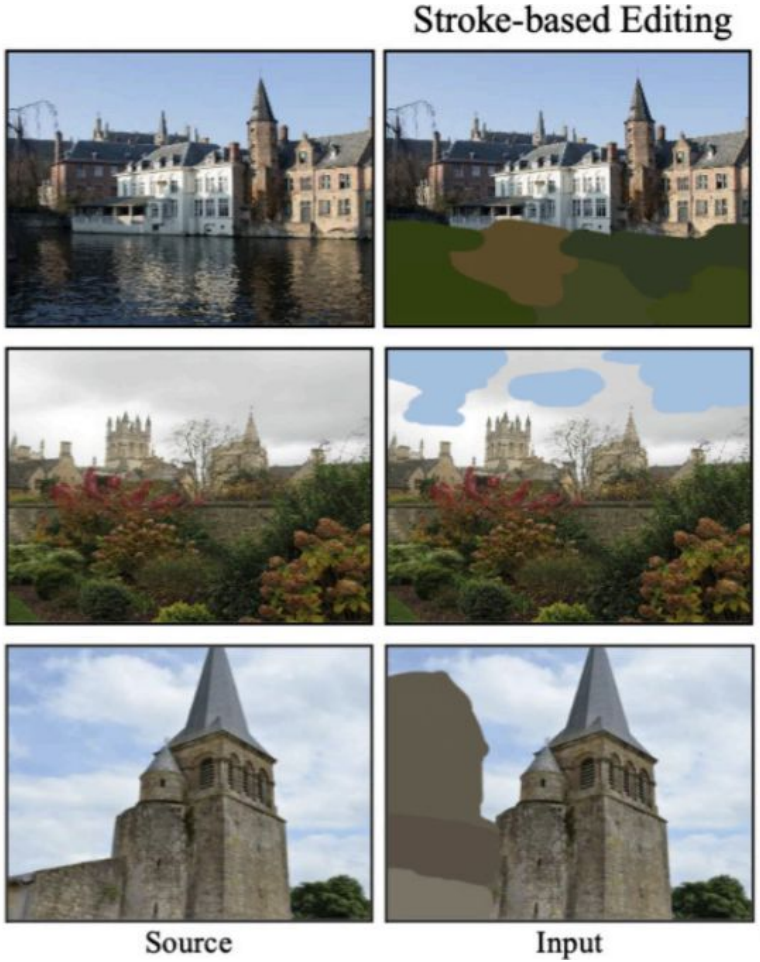


SDEdit

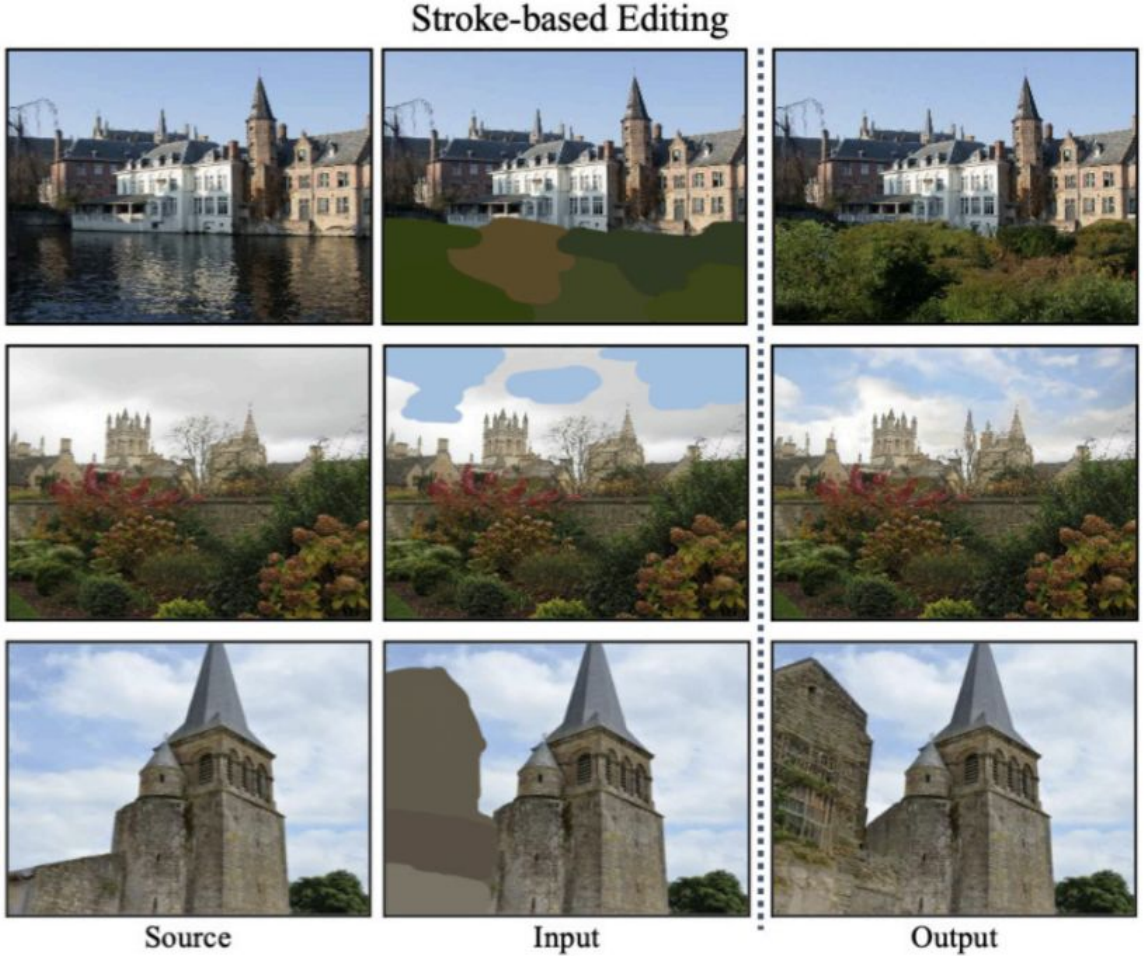
Stroke-based Editing

Source

SDEdit

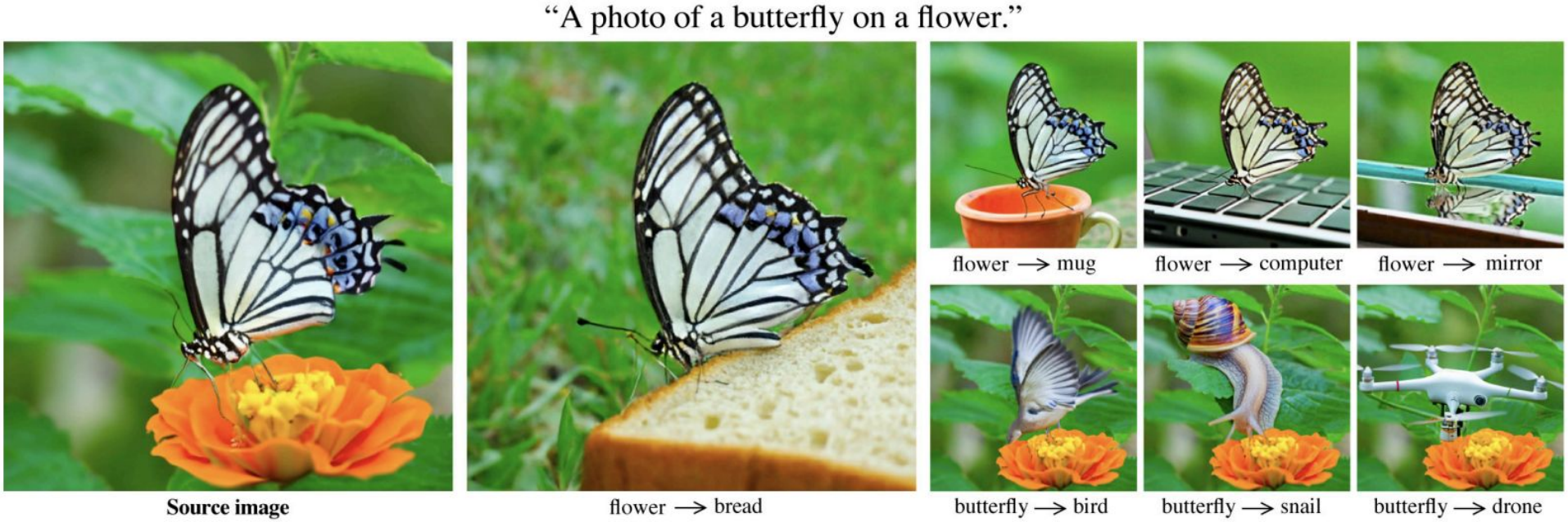


SDEdit



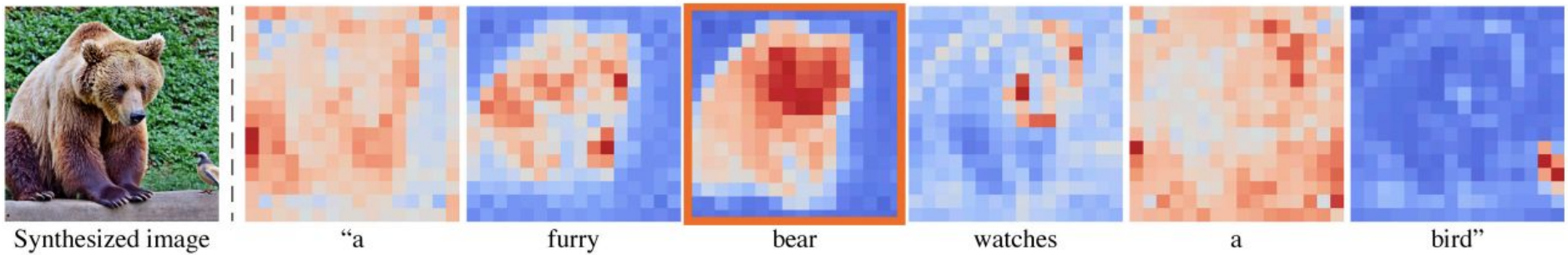
Prompt-to-Prompt

Prompt-to-Prompt



Prompt-to-Prompt

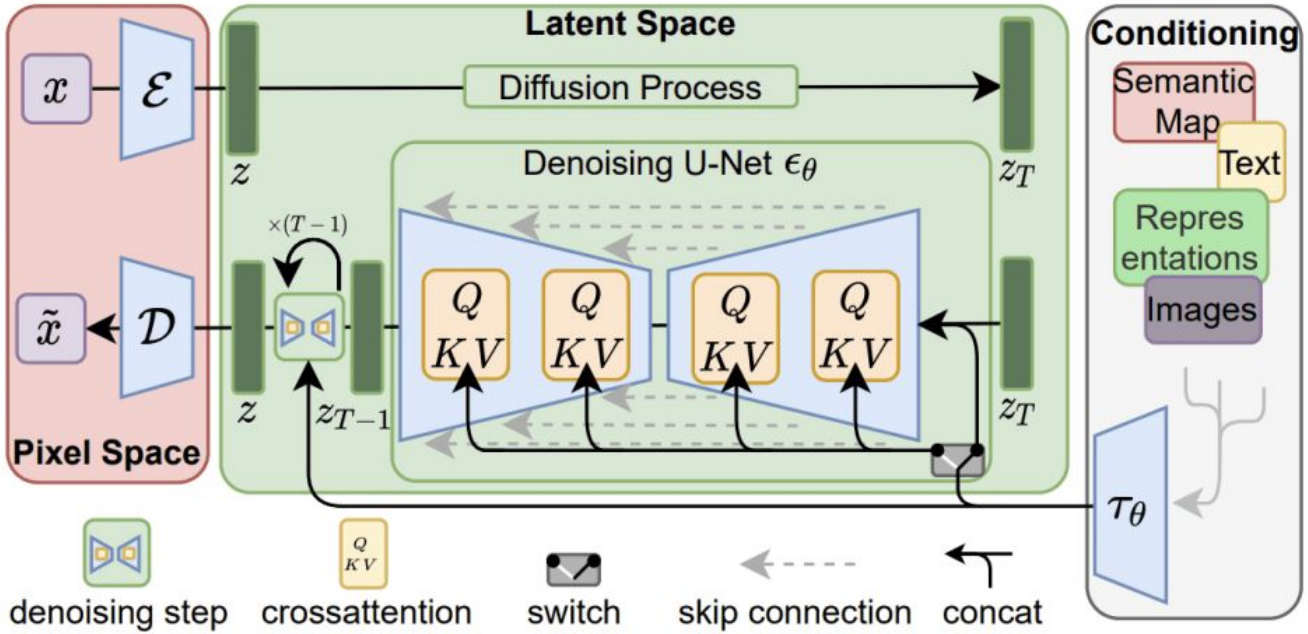
(High Level) Idea: Features inside diffusion models encode very high level information such as: style, content, and structure.



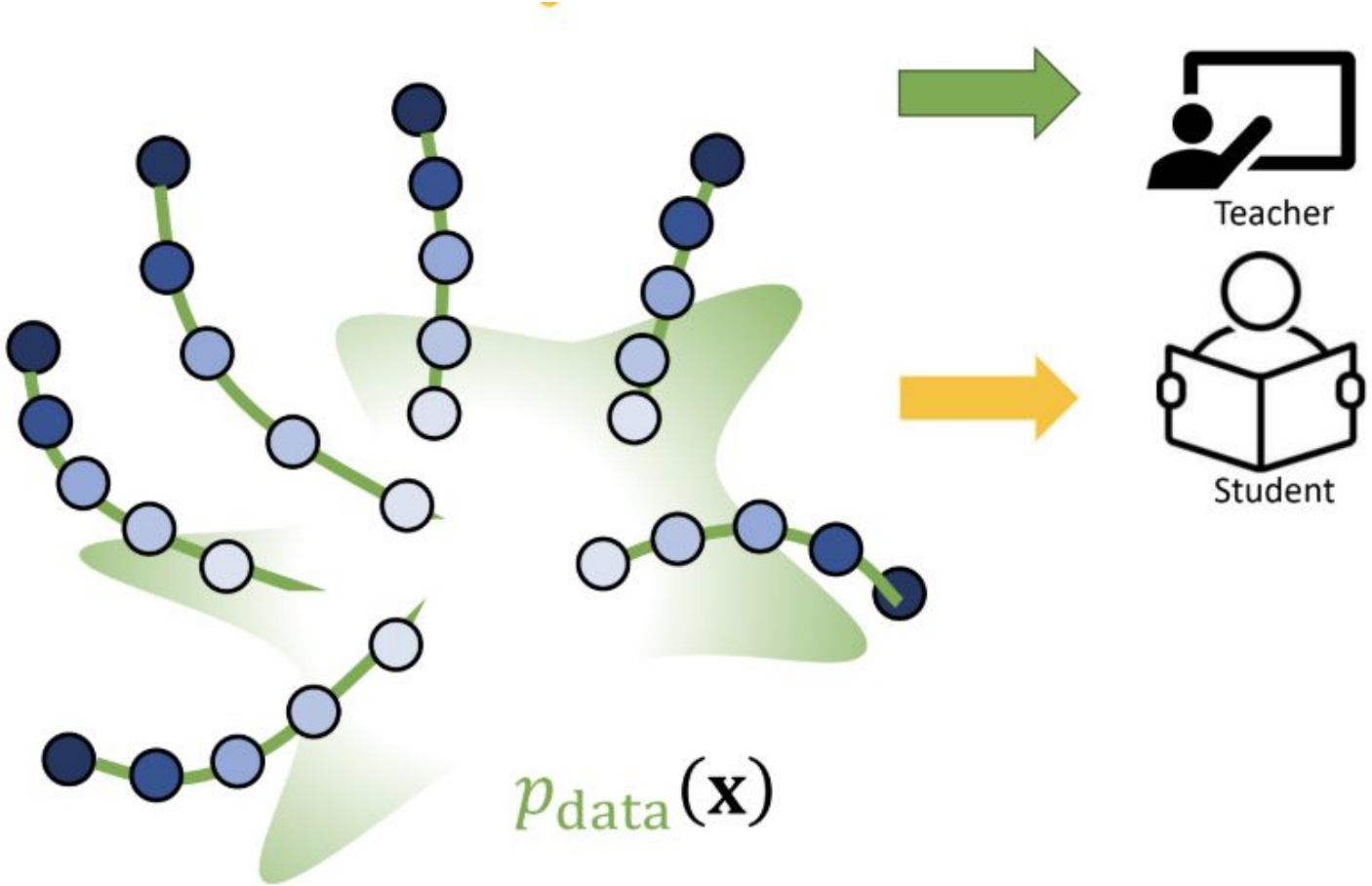
Reuse (copy and paste) the features from the previous prompt

Latent Diffusion

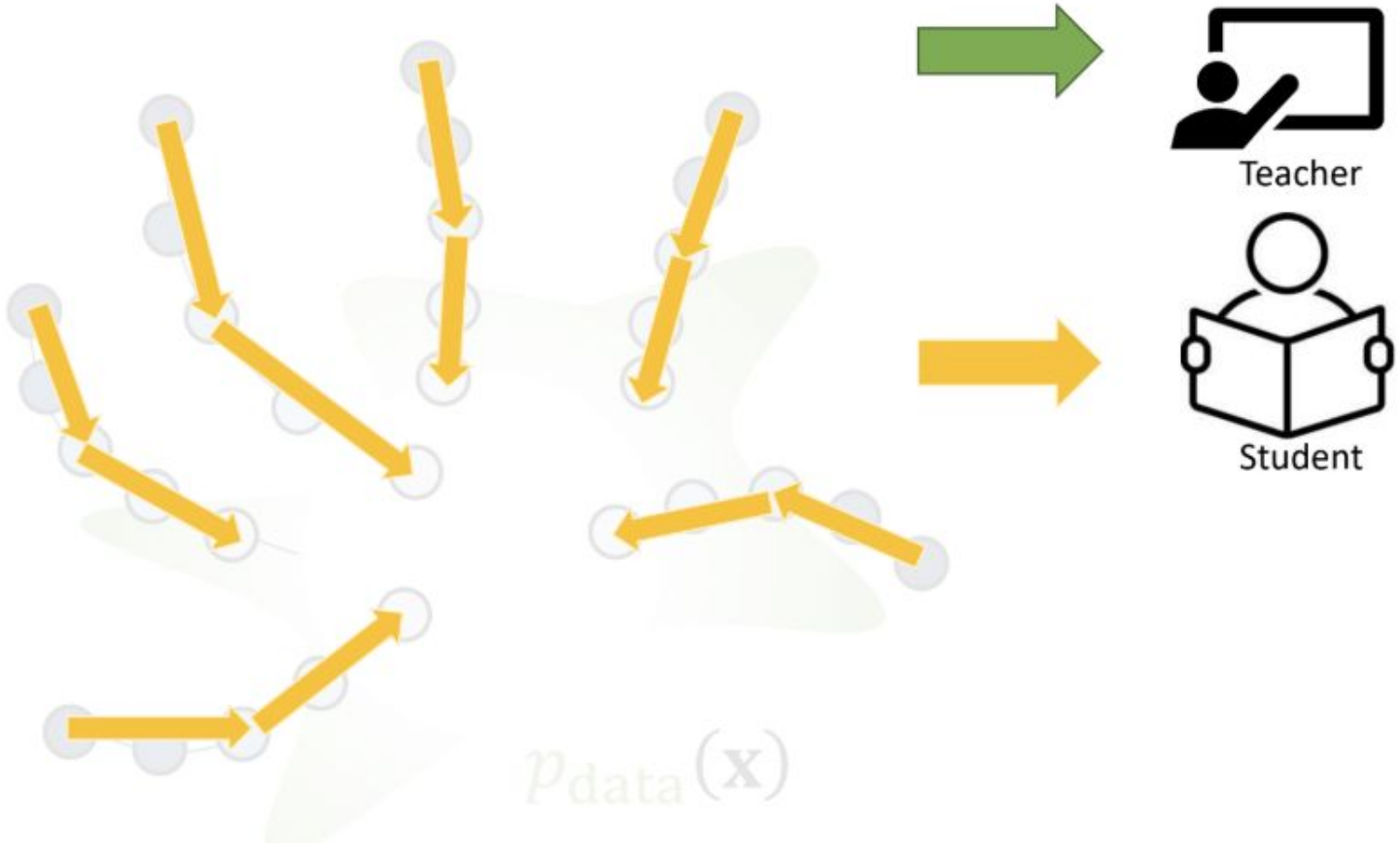
Even for a 64x64 image, running a diffusion model for a large number of steps will be very expensive. To combat this problem, we can train an autoencoder, and do the diffusion in the latent space:



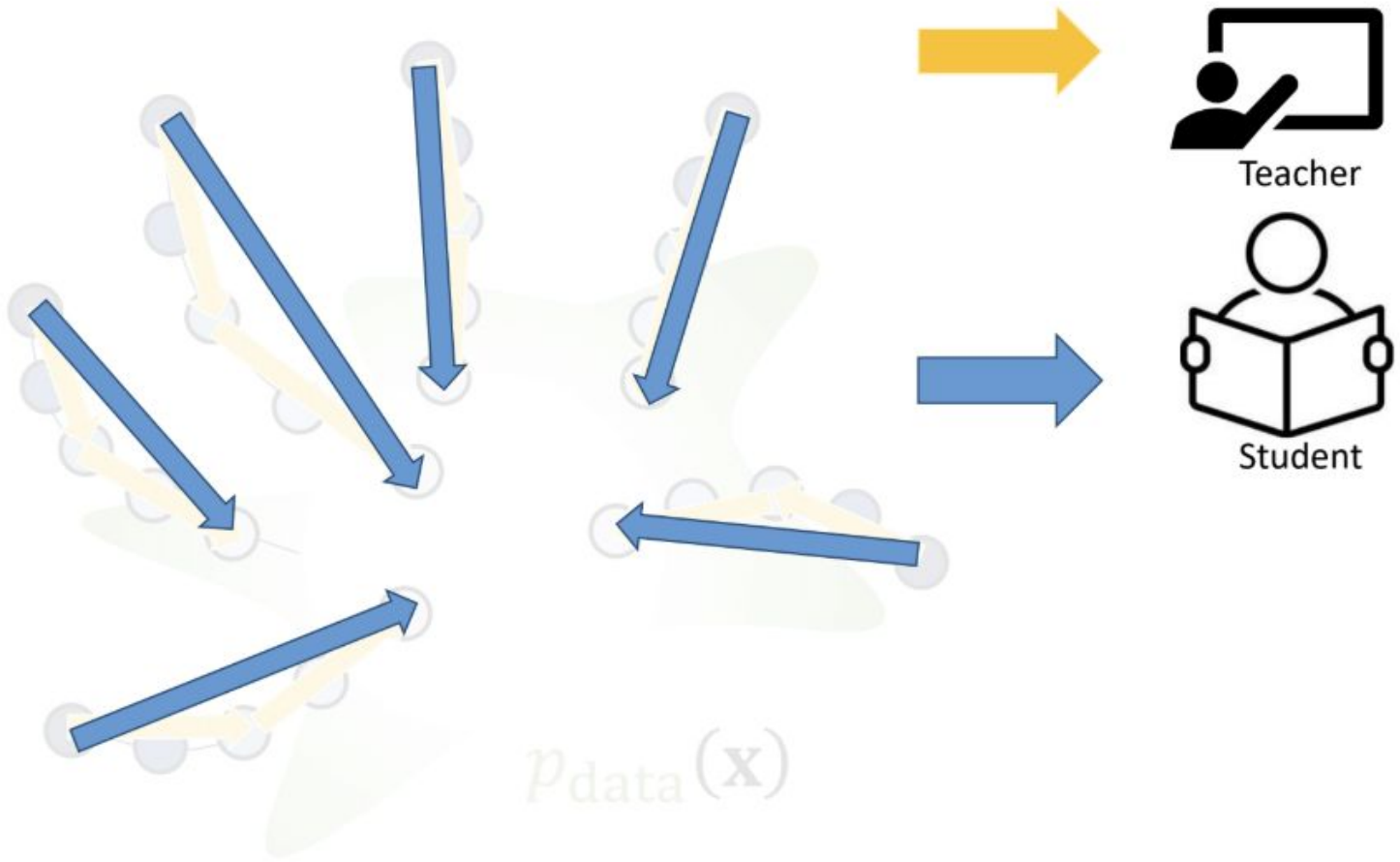
Progressive Distillation



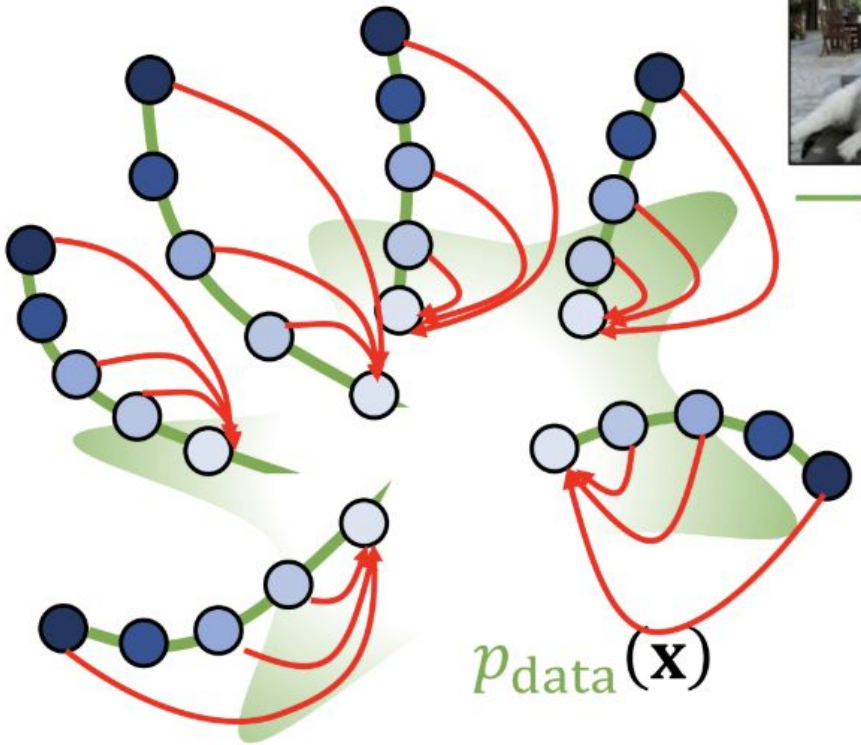
Progressive Distillation

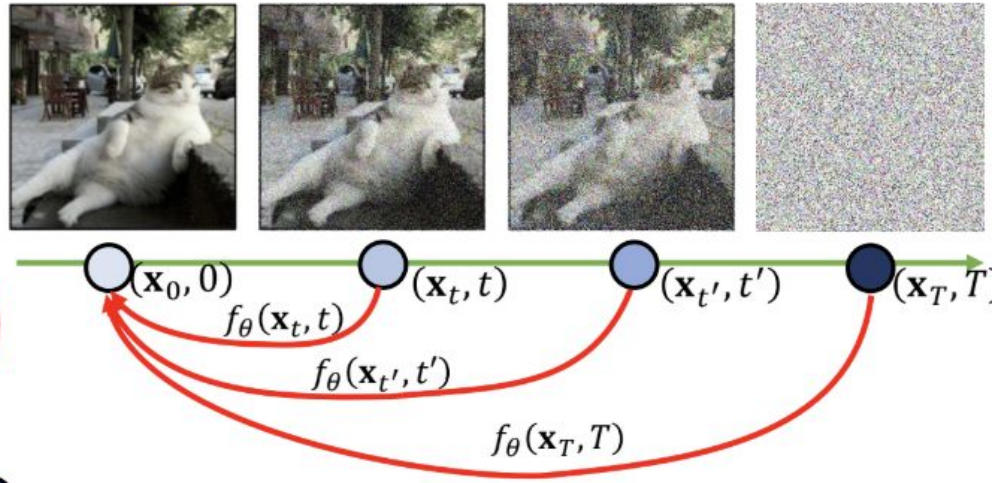


Progressive Distillation



Consistency Models



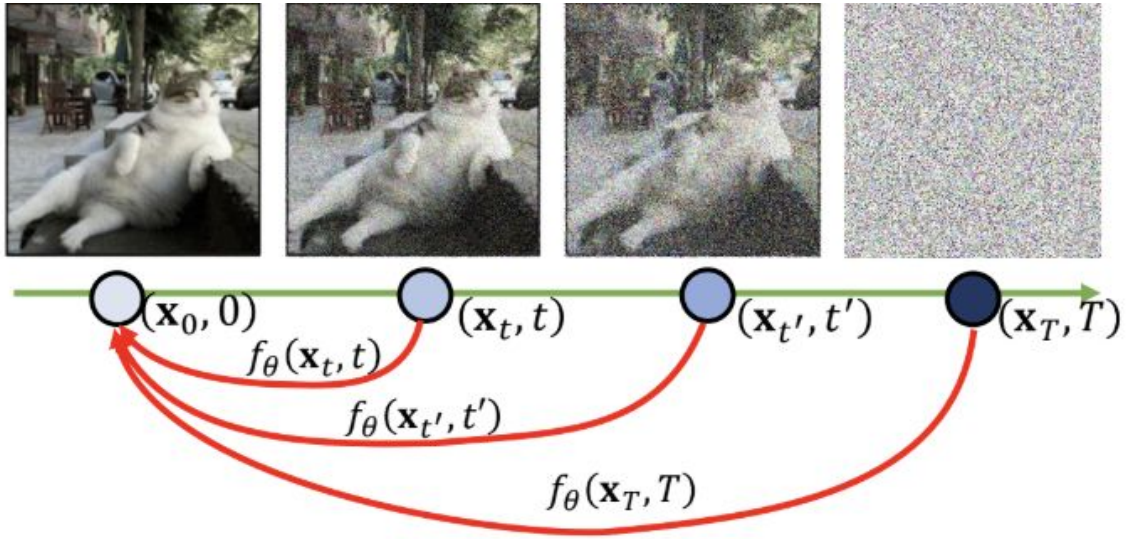


$$\min_\theta d\Big(f_{EMA}(\mathbf{x}_t, t), f_\theta(\mathbf{x}_{t'}, t')\Big)$$

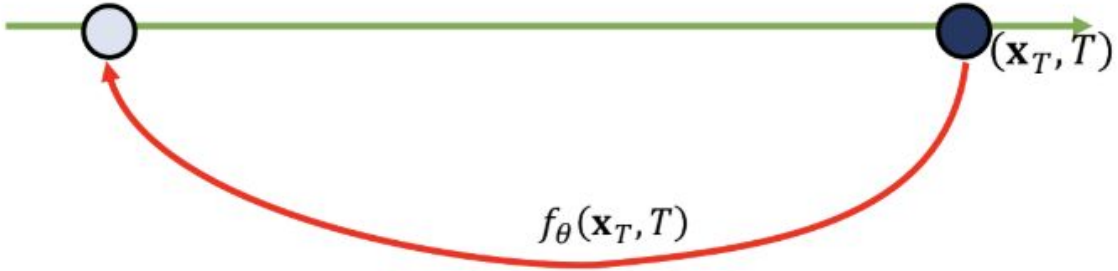
target network (teacher)

online network (student)

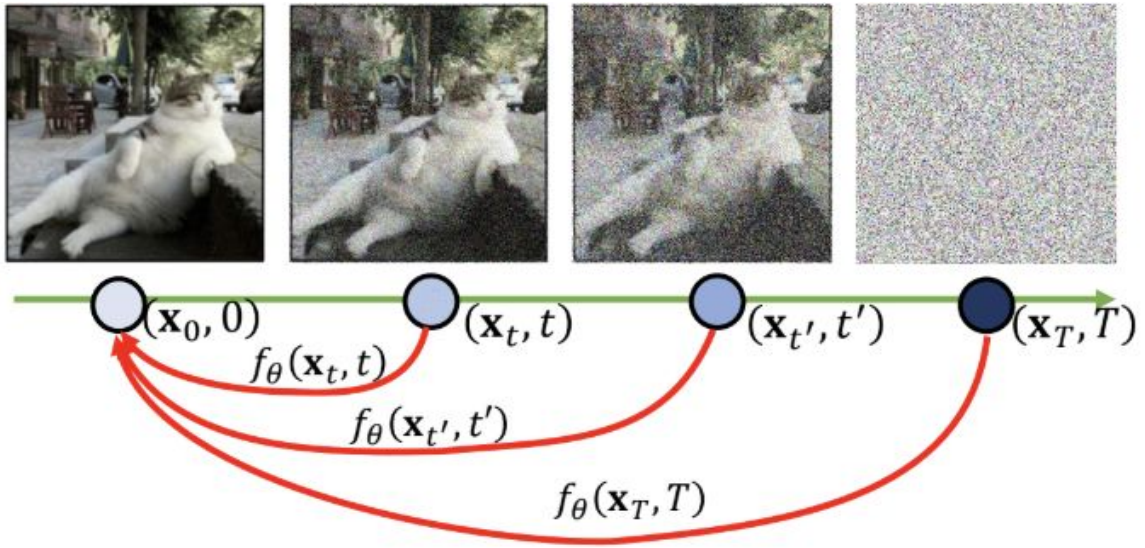
Consistency Models



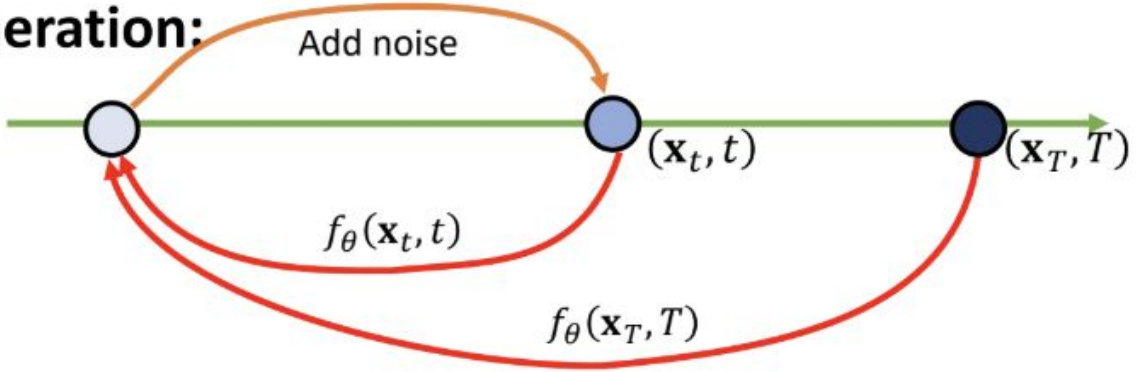
Single-step generation:



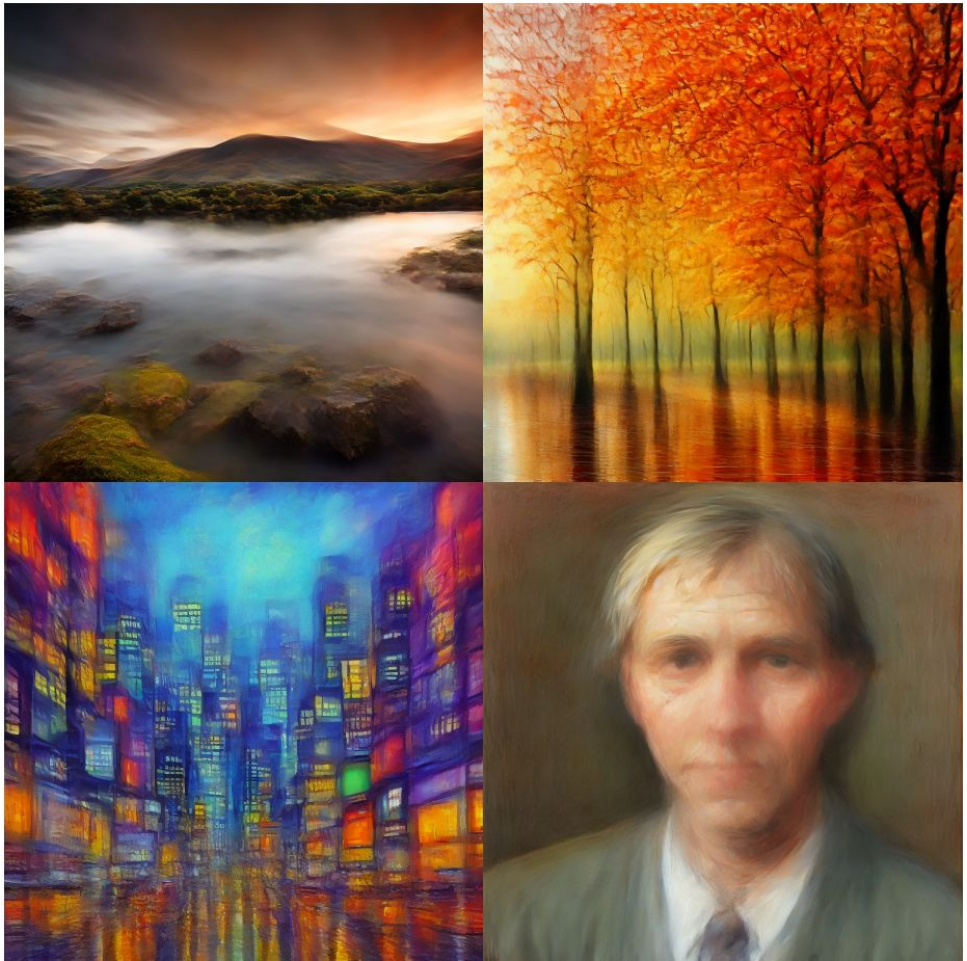
Consistency Models



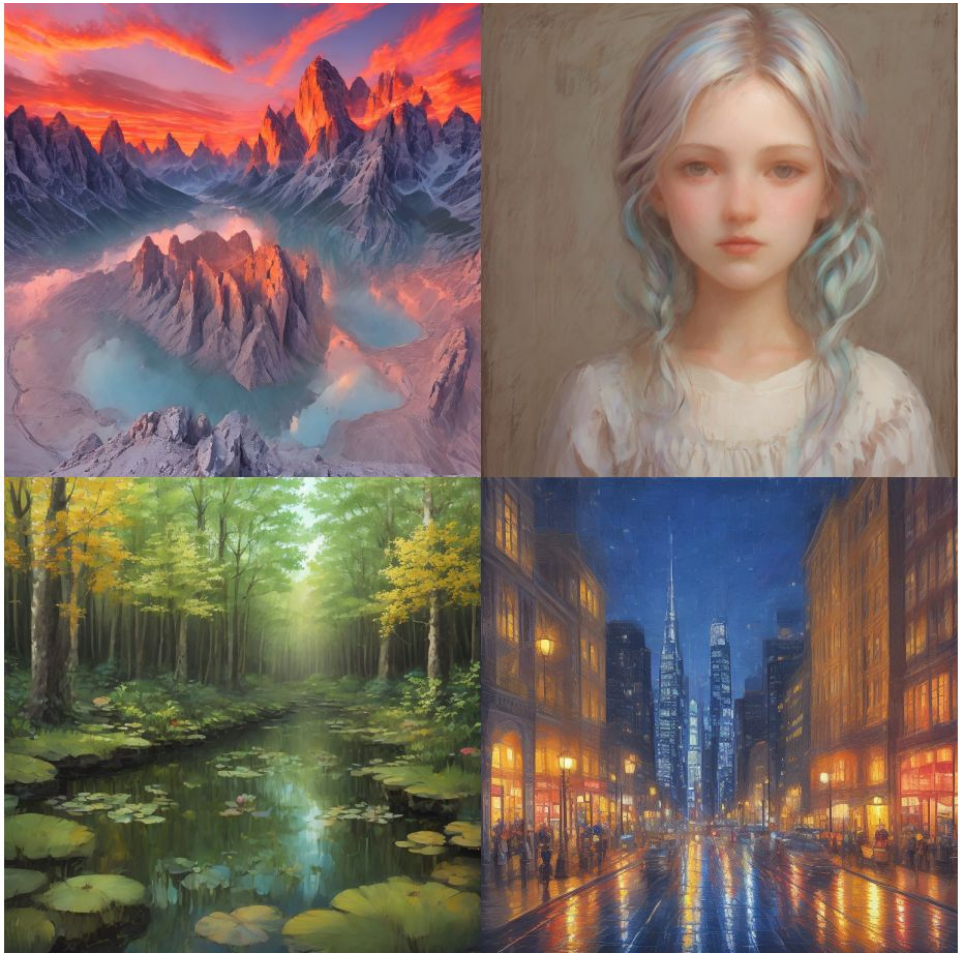
Multi-step generation:



Latent Consistency Models (1-Step Inference)

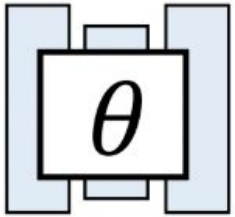


Latent Consistency Models (2-Steps Inference)



Latent Consistency Models (2-Steps Inference)

Low-rank Adaptation (LoRA):

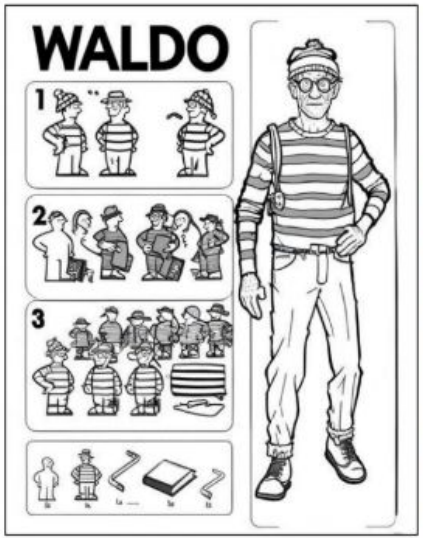


U-Net

Pretrained diffusion model

PixelArt

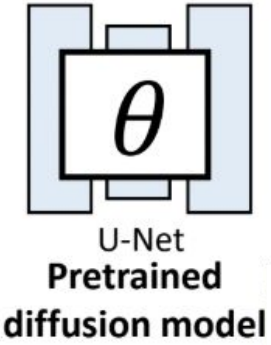
Lego



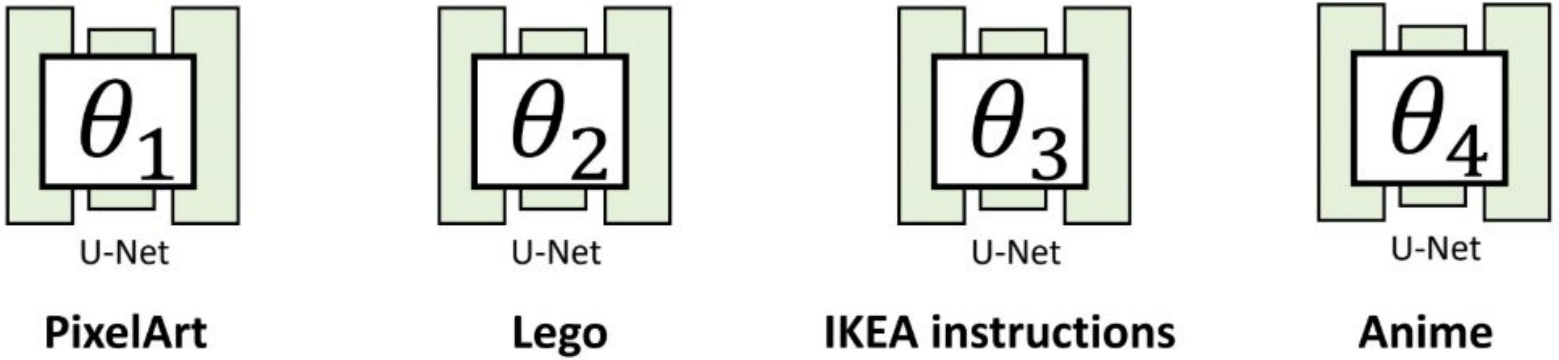
IKEA instructions

Anime

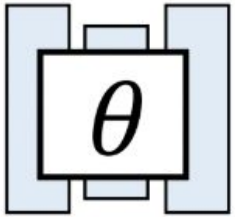
Low-rank Adaptation (LoRA):



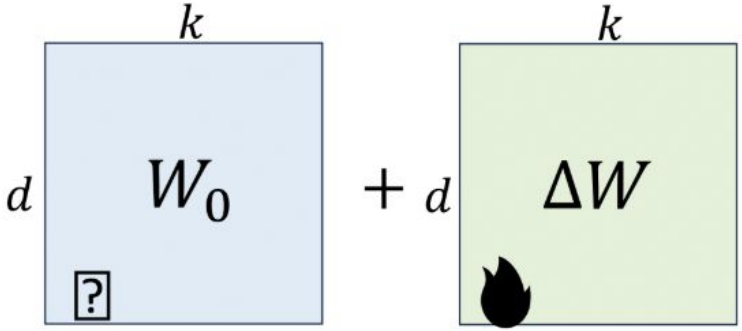
- Computationally Expensive
- High Storage Requirement
- Slower Training & Inference
- Inefficient for Multiple Tasks

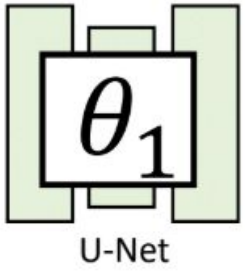


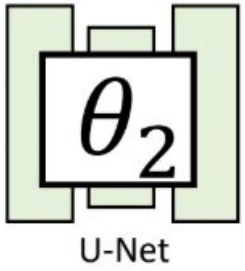
Low-rank Adaptation (LoRA):

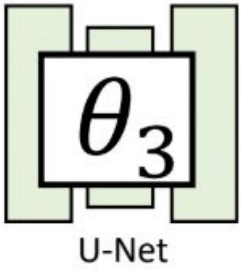


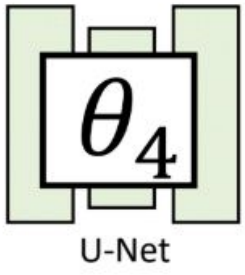
- Finetuning only cross-attention layers



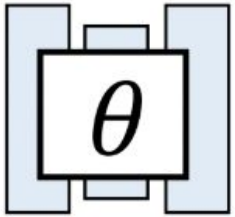




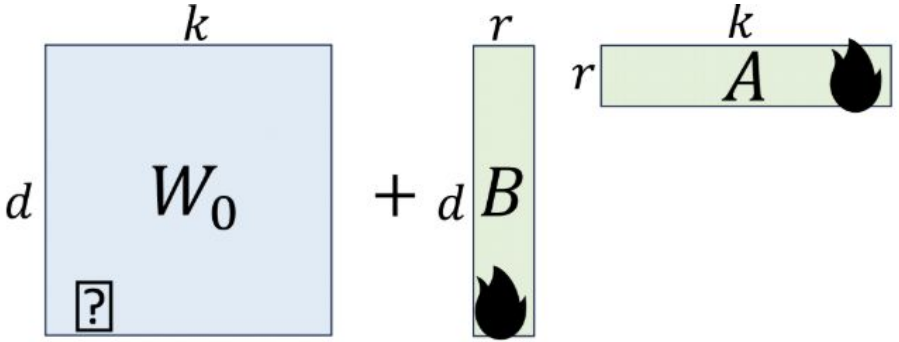


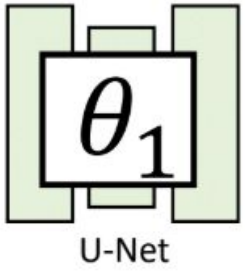


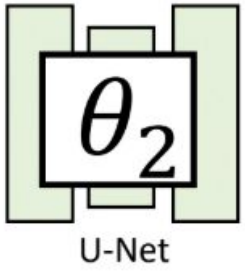
Low-rank Adaptation (LoRA):

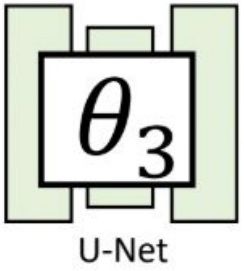


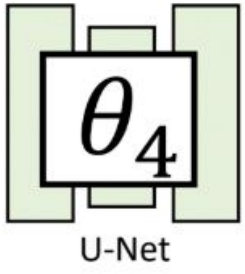
- Finetuning only cross-attention layers



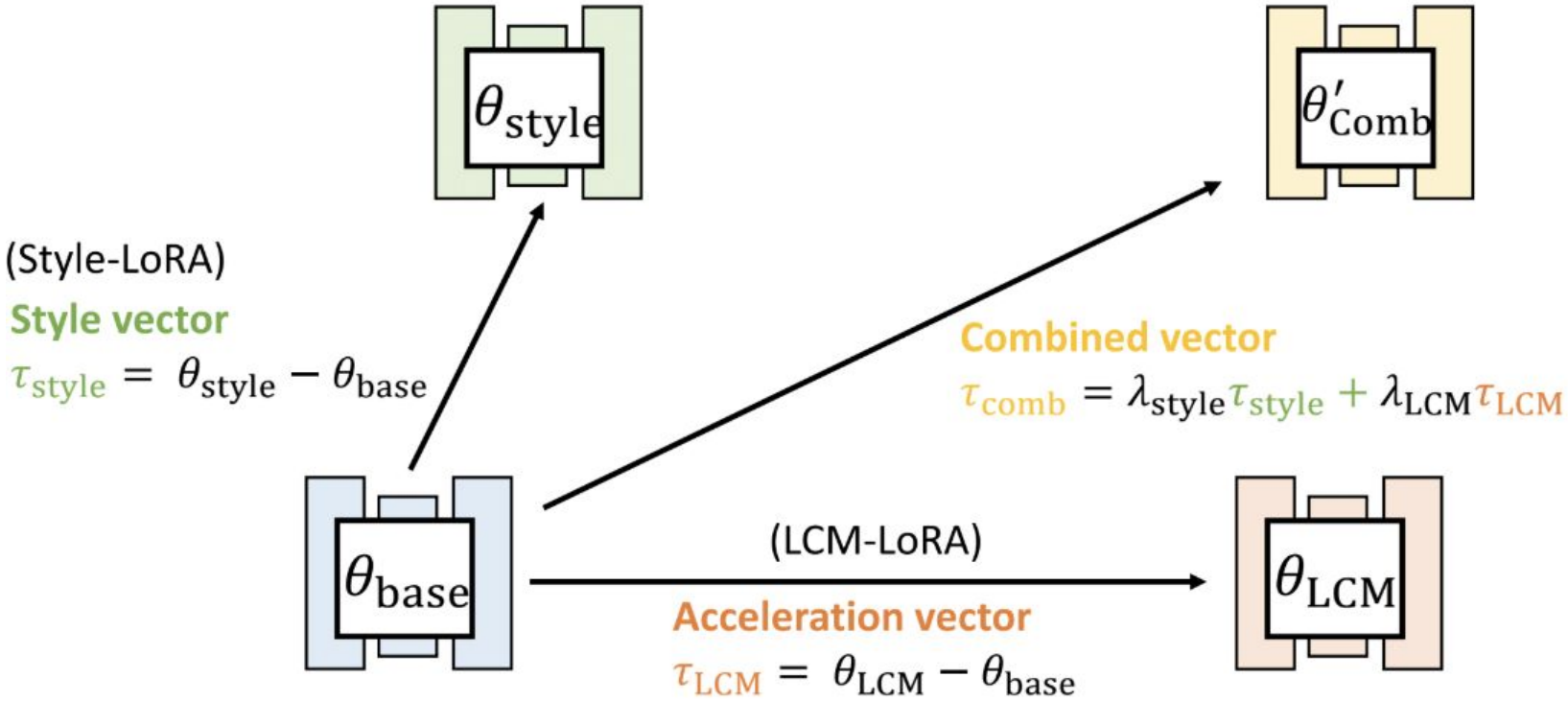






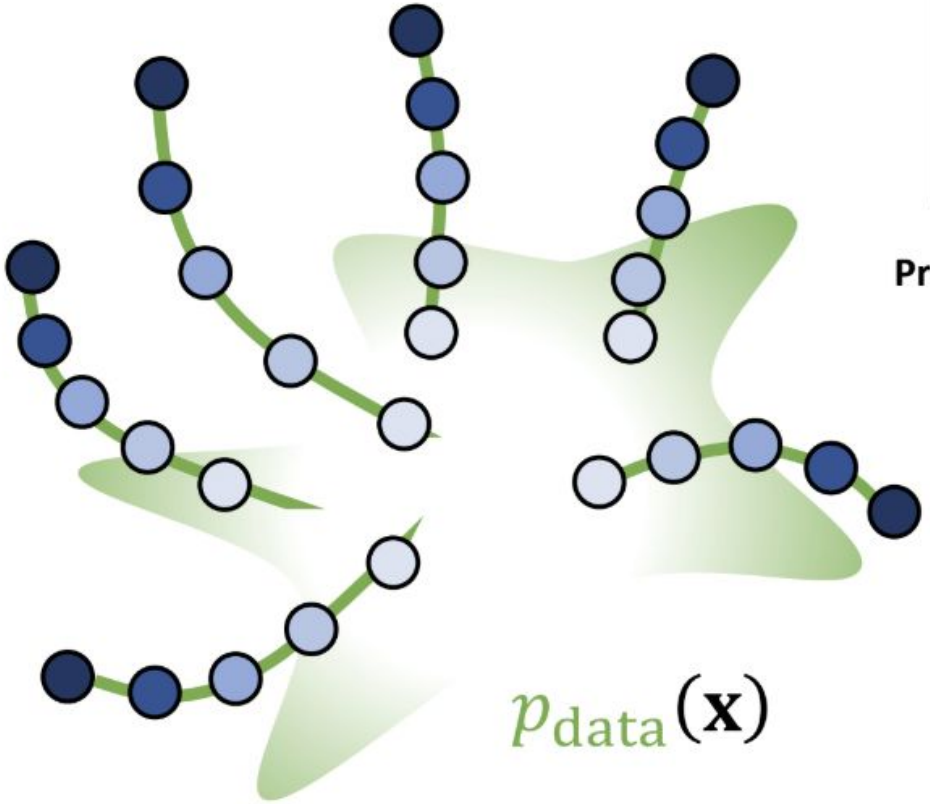


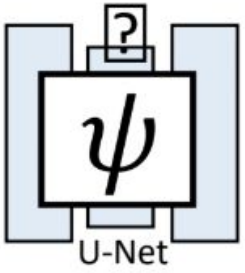
Latent Consistency Models + LoRA



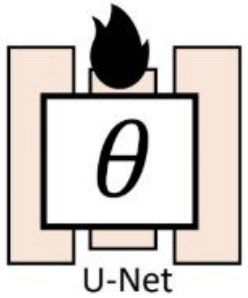
Latent Consistency Models + LoRA

Adversarial Diffusion Distillation



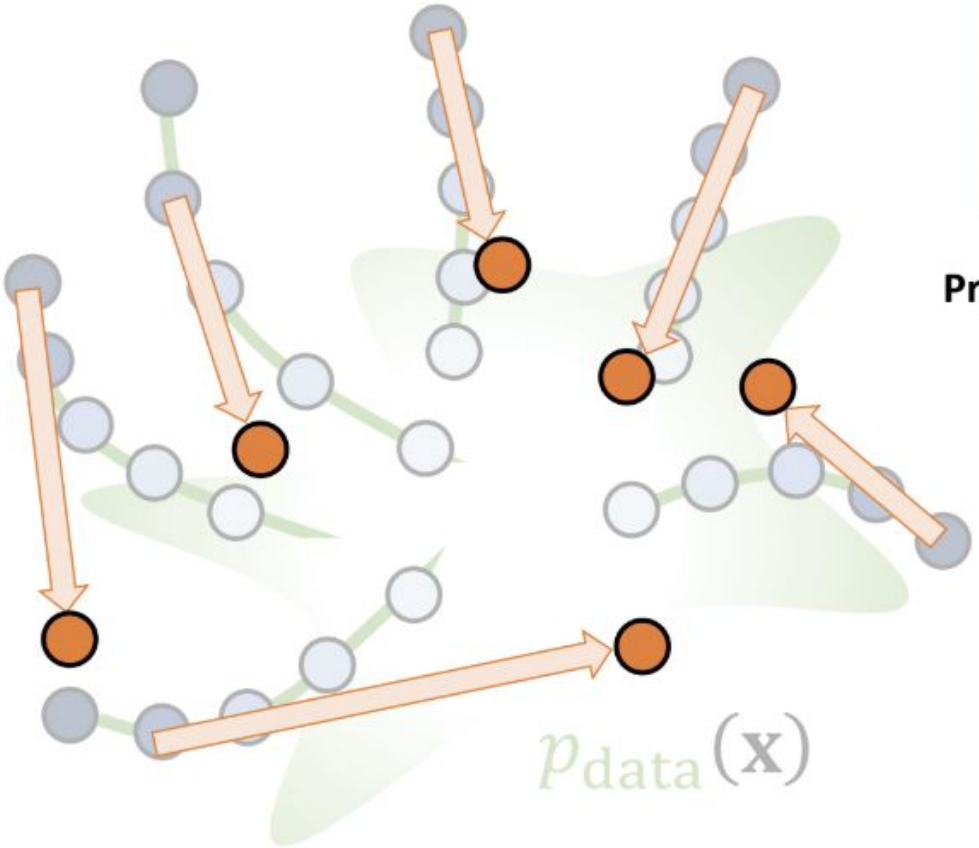


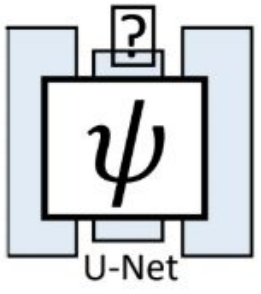
Pretrained model



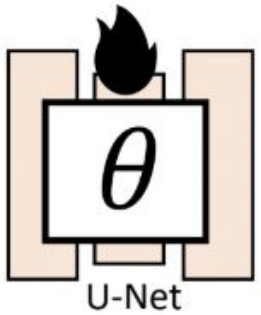
Student model

Adversarial Diffusion Distillation



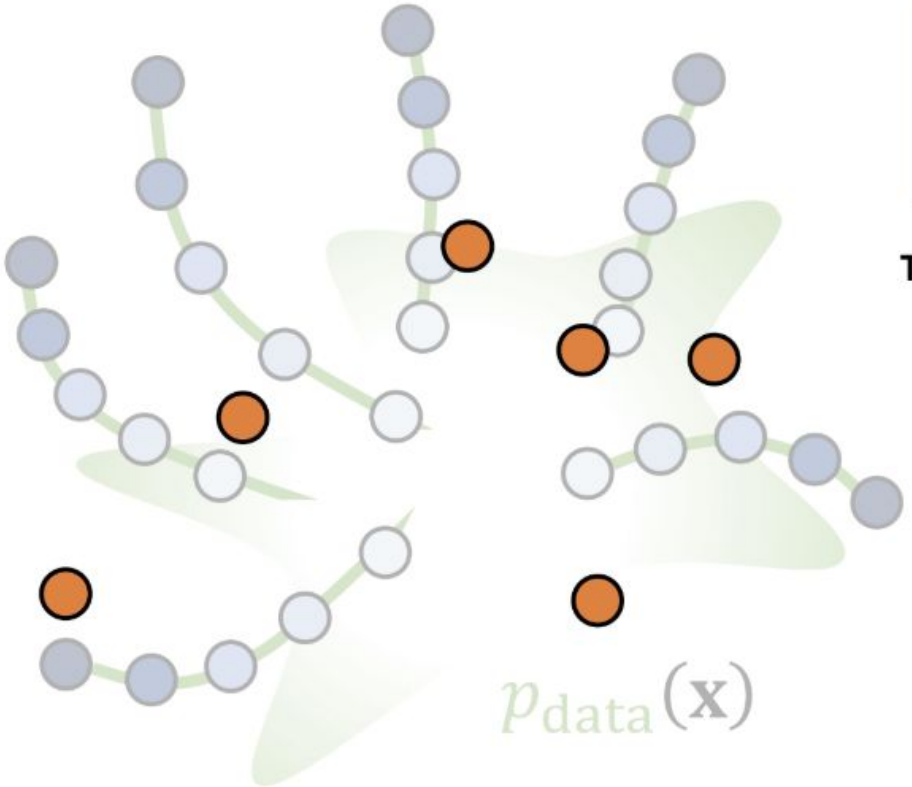


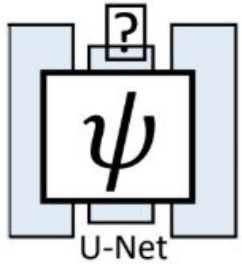
Pretrained model



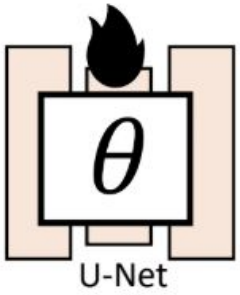
Student model

Adversarial Diffusion Distillation

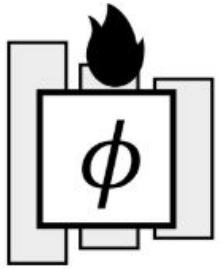




Teacher model



Student model



Discriminator

Problem:

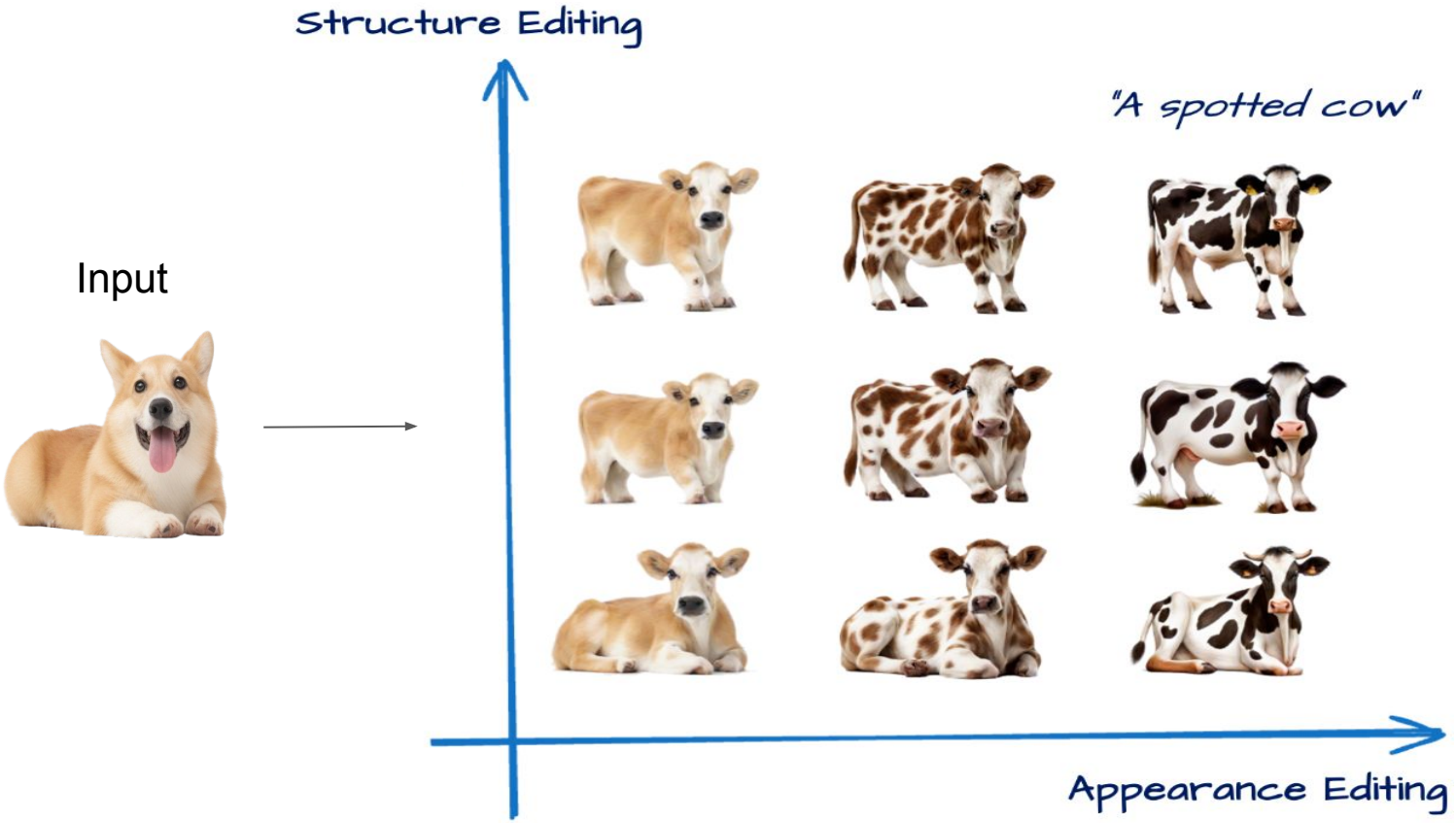
Denoised image predicted by the teacher: blurry

Idea:

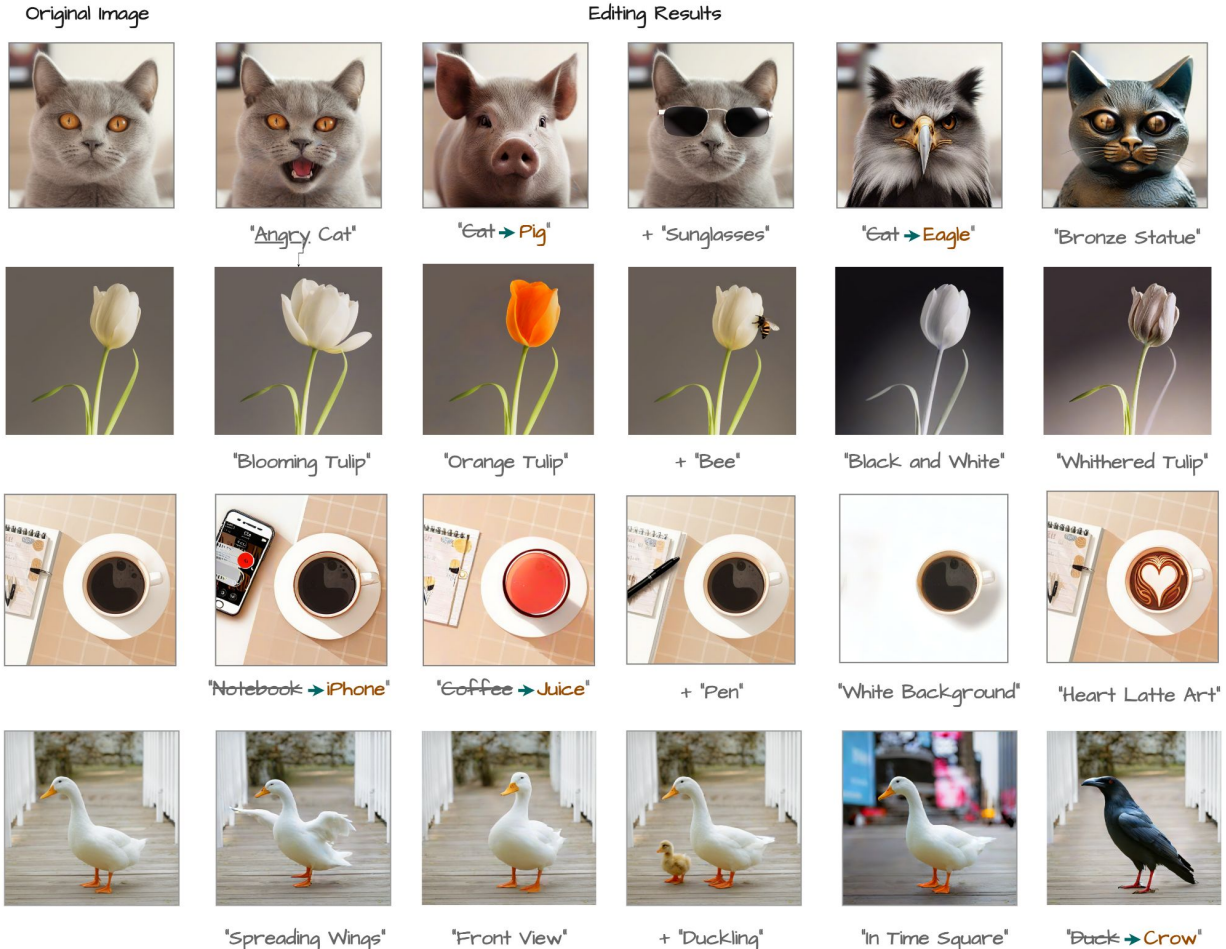
Use both adversarial loss and score distillation loss

Adversarial Diffusion Distillation

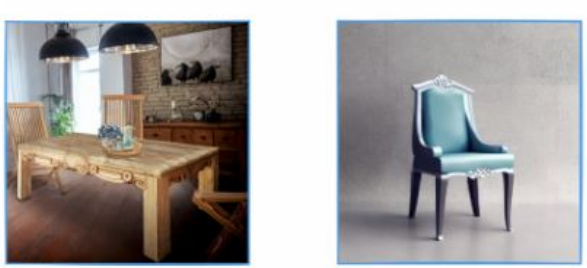
Cora: Correspondence-aware image editing



Cora: Correspondence-aware image editing

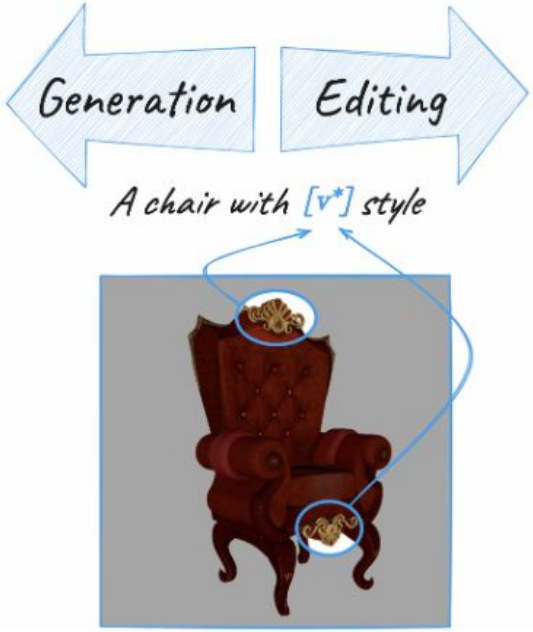


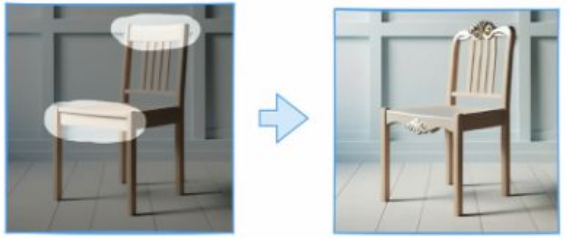
CLiC: Concept Learning in Context



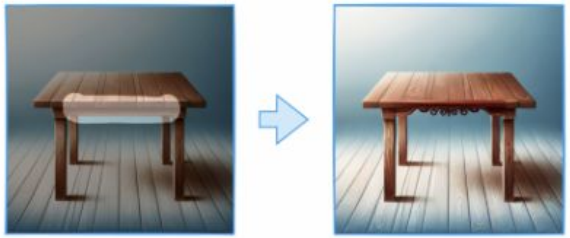
A table/chair with [v*] style

A bed/desk with [v*] style



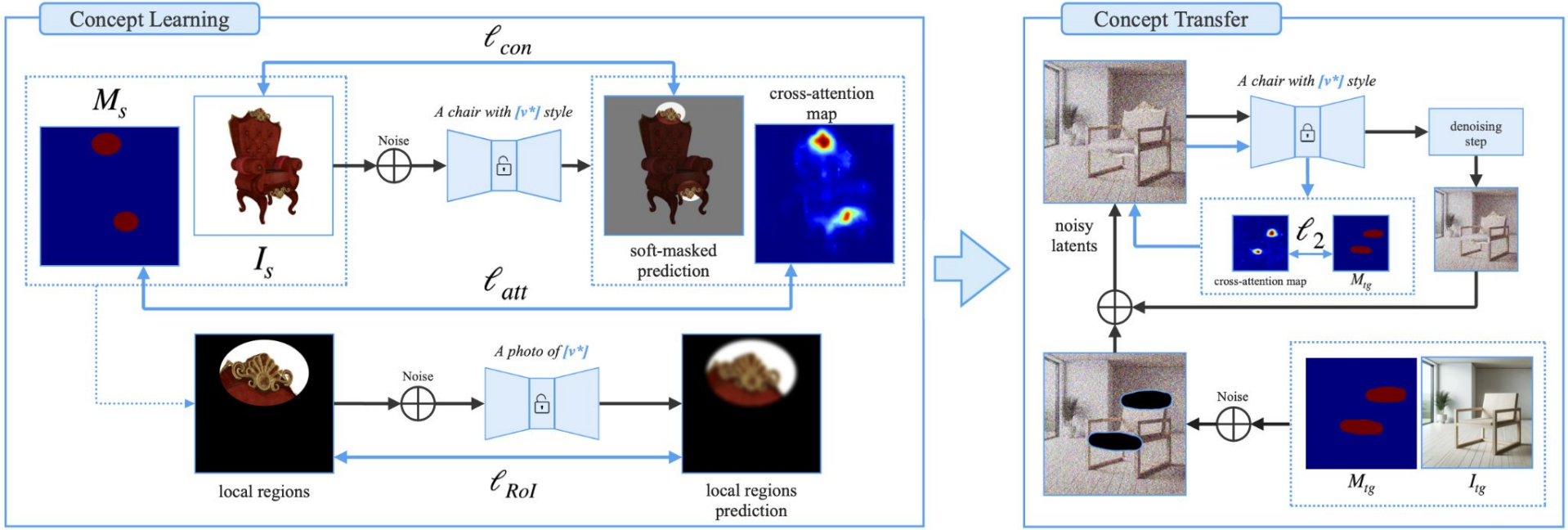


A chair with [v*] style



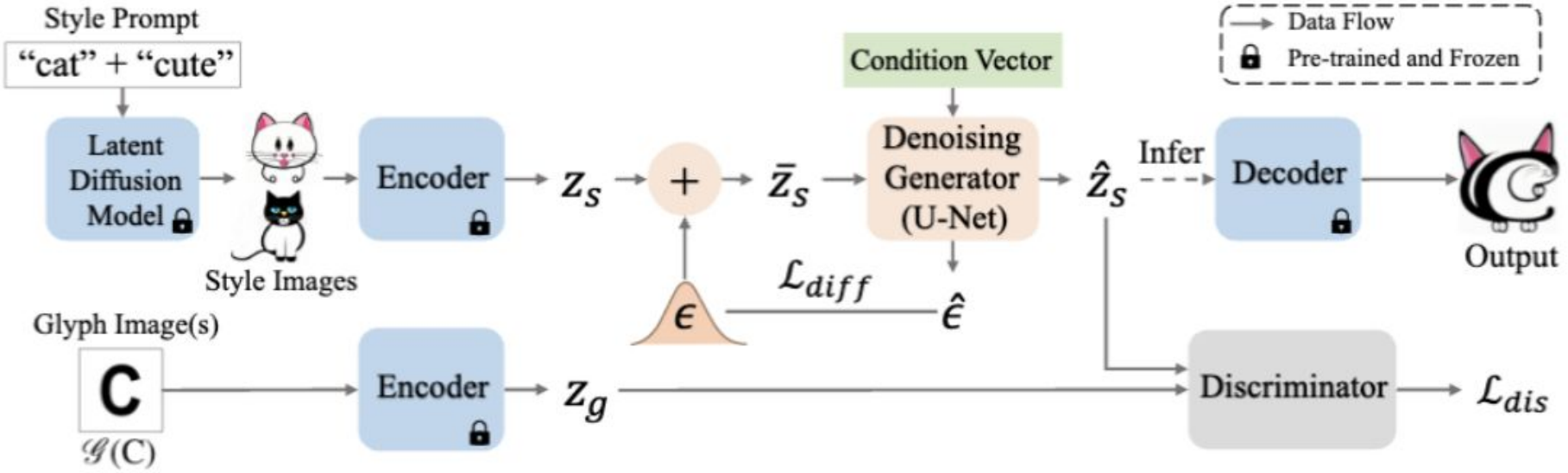
A table with [v*] style

CLiC: Concept Learning in Context



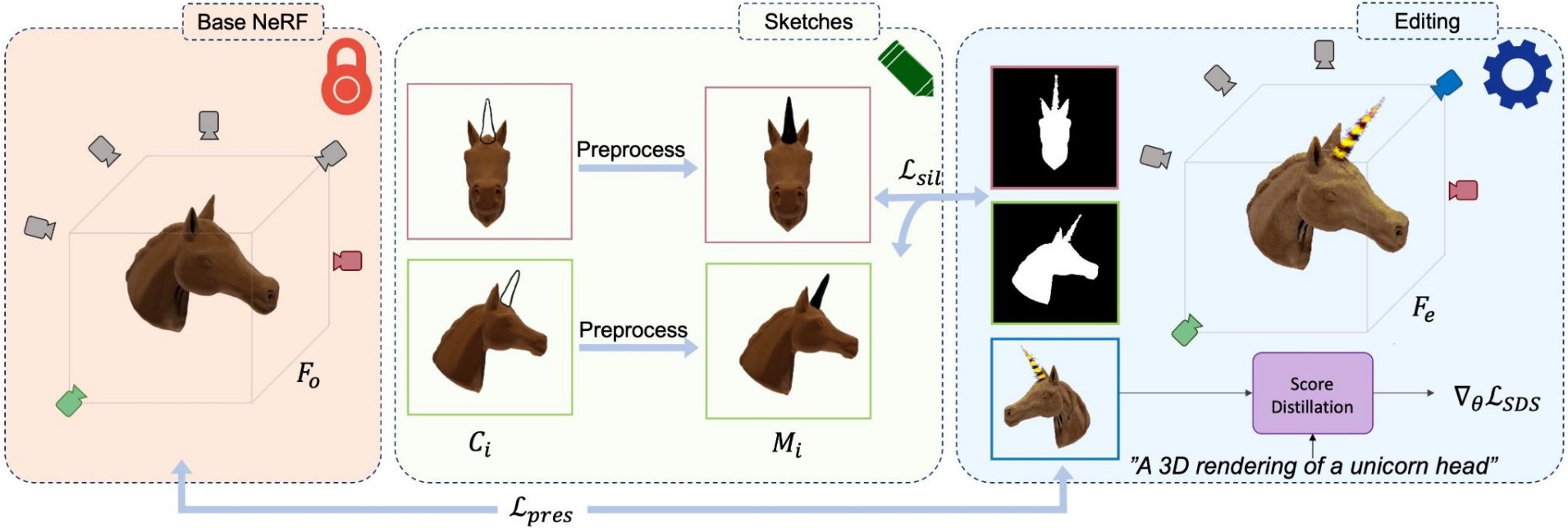
DS-Fusion: Artistic Typography via Discriminated and Stylized Diffusion

DS-Fusion: Artistic Typography via Discriminated and Stylized Diffusion

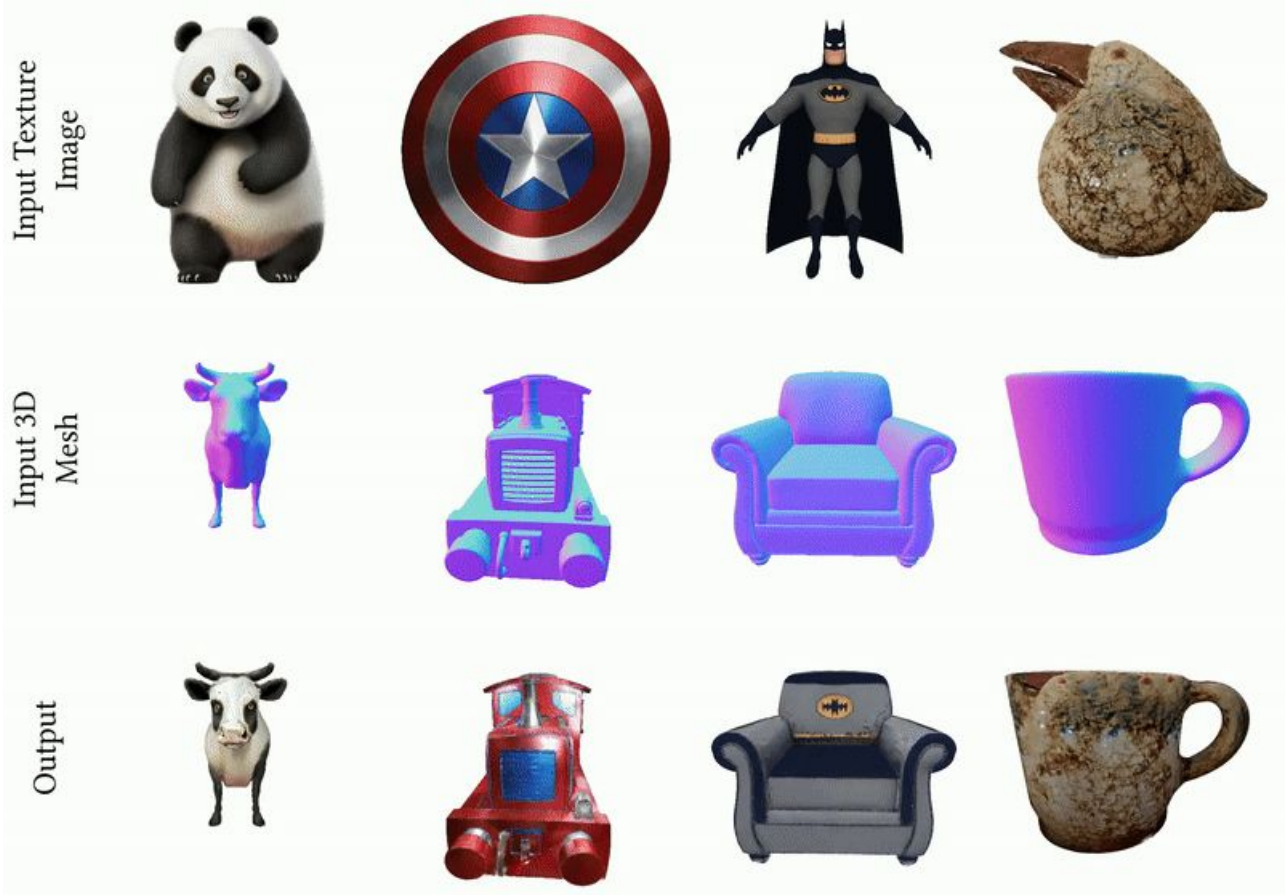


SKED: Sketch-guided Text-based 3D Editing

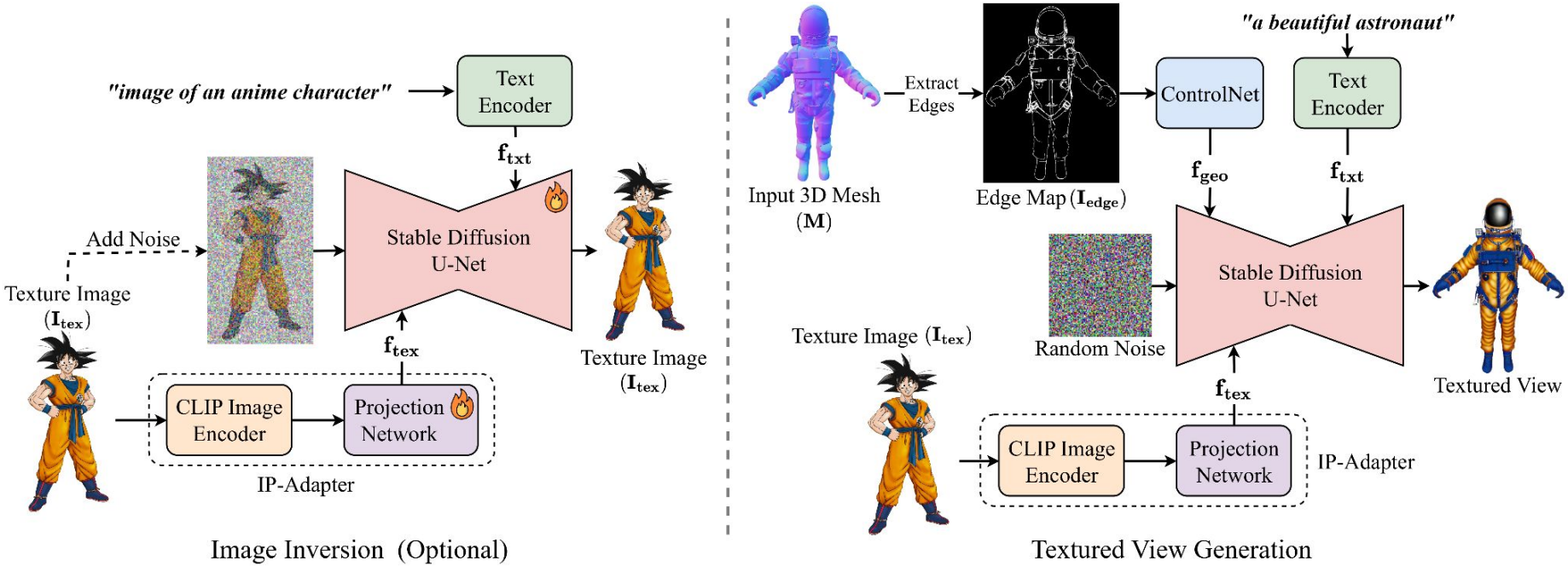
SKED: Sketch-guided Text-based 3D Editing



EASI-Tex: Edge-Aware Mesh Texturing from Single Image



EASI-Tex: Edge-Aware Mesh Texturing from Single Image



Reference

https://www.assemblyai.com/blog/diffusion-models-for-machine-learning-introduction/#references

https://yang-song.net/blog/2021/score/

https://www.eecs.umich.edu/courses/eecs442-ahowens/fa23/slides/lec11-diffusion.pdf

https://www.dropbox.com/scl/fi/rjfr15oq1qoosfmldetvc/04_speed.pptx?rlkey=ymg7phidx9693cl0cwoduzwrq&e=1&dl=0